UNITED STATES DISTRICT COURT
WESTERN DISTRICT OF WASHINGTON
AT SEATTLE

MICROSOFT CORPORATION, a Washington Corporation Plaintiff, v. SALESFORCE.COM, INC., a Delaware Corporation Defendants.	Case No: **COMPLAINT FOR PATENT INFRINGEMENT** **JURY DEMAND**

Plaintiff Microsoft Corporation ("Microsoft") for its Complaint For Patent Infringement against Defendant Salesforce.com, Inc. ("Defendant"), alleges as follows:

PARTIES

1. Plaintiff Microsoft Corporation is a Washington corporation having its principal place of business at One Microsoft Way, Redmond, Washington 98052.

2. Founded in 1975, Microsoft is a worldwide leader in computer software, services, and solutions for businesses and consumers. Since 1979, Microsoft has been headquartered in the Seattle, Washington metropolitan area. Microsoft currently employs nearly 40,000 people in

Microsoft Corporation
1 Microsoft Way
Redmond, Washington 98052
Telephone: 425-882-8080

the Puget Sound area and occupies nearly 8 million square feet of facilities at its Redmond campus.

3. Microsoft has a long history of technical innovation in the software and hardware products it develops and distributes. These software products include operating systems for servers, personal computers, embedded devices, smartphones, PDAs, and other intelligent devices; "cloud" computing platforms, such as Windows Azure; customer relationship management software, such as Microsoft Dynamics CRM and Microsoft Dynamics ERP; server applications for distributed computing environments; various web applications and services; information worker productivity applications; business solution applications; high-performance computing applications; and software development tools.

4. On information and belief, Defendant Salesforce.com, Inc. is a United States corporation organized and existing under the laws of Delaware having a principal place of business at The Landmark at One Market Street, Suite 300, San Francisco, CA 94105.

5. On information and belief, Defendant is in the business of developing and providing customer relationship management (CRM) software as a service over the Internet. On information and belief, Defendant offers this CRM "software as a service" (SAAS) worldwide, including in the United States, via its websites and servers, which are located throughout the United States. On information and belief, Defendant does business within the Western District of Washington.

JURISDICTION AND VENUE

6. This is an action for patent infringement arising under the patent laws of the United States, Title 35, United States Code.

7. This Court has subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

8. Venue is proper in this district pursuant to 28 U.S.C. §§ 1391(b), 1391(c) and 1400(b). On information and belief, Defendant is subject to this Court's personal jurisdiction,

Microsoft Corporation
1 Microsoft Way
Redmond, Washington 98052
Telephone: 425-882-8080

consistent with the principles of due process and the Washington Long Arm Statute, because Defendant offers its services for sale in the Western District of Washington, has transacted business in this District, and/or has committed and/or induced acts of patent infringement in this District.

PATENT INFRINGEMENT COUNTS

9. Microsoft is the owner of all right, title, and interest in U.S. Patent Nos. 7,251,653; 5,742,768; 5,644,737; 6,263,352; 6,122,558; 6,542,164; 6,281,879; 845,077; and 5,941,947 (collectively, "the Microsoft patents-in-suit"), which the Defendant is infringing and/or inducing others to infringe by, among other things, making, using, making available for another's use, offering to license or licensing in the United States, offering to sell or selling in the United States, or importing into the United States, products or processes that practice inventions claimed in the Microsoft patents-in-suit.

10. The Defendant has profited through infringement of the Microsoft patents-in-suit. As a result of the Defendant's unlawful infringement of the Microsoft patents-in-suit, Microsoft has suffered and will continue to suffer damage. Microsoft is entitled to recover from the Defendant the damages suffered by Microsoft as a result of the Defendant's unlawful acts.

11. On information and belief, Defendant's infringement of the Microsoft patents-in-suit is willful and deliberate, entitling Microsoft to enhanced damages and reasonable attorney fees and costs. Microsoft has provided Defendant notice of its infringement through, *inter alia*, service of this complaint and prior communications between the parties.

12. On information and belief, the Defendant intends to continue its unlawful infringing activity, and Microsoft continues to and will continue to suffer irreparable harm—for which there is no adequate remedy at law—from such unlawful infringing activity unless Defendant is enjoined by this Court.

Microsoft Corporation
1 Microsoft Way
Redmond, Washington 98052
Telephone: 425-882-8080

COUNT I

INFRINGEMENT OF U.S. PATENT NO. 7,251,653

13. Microsoft realleges and incorporates by reference the allegations set forth in paragraphs 1-12.

14. Microsoft is the owner of all right, title, and interest in U.S. Patent No. 7,251,653 ("the '653 patent"), entitled "Method and system for mapping between logical data and physical data," duly and properly issued by the U.S. Patent and Trademark Office on July 31, 2007. A copy of the '653 patent is attached as Exhibit A.

15. The Defendant has been and/or is directly infringing the '653 patent by, among other things, making, using, offering to license or licensing in the United States, offering to sell or selling in the United States, products and/or services, including various web applications and services and the hardware and software running these applications and services, that embody or incorporate, or the operation of which otherwise practices, one or more claims of the '653 patent.

COUNT II

INFRINGEMENT OF U.S. PATENT NO. 5,742,768

16. Microsoft realleges and incorporates by reference the allegations set forth in paragraphs 1-12.

17. Microsoft is the owner of all right, title, and interest in U.S. Patent No. 5,742,768 ("the '768 patent"), entitled "System and method for providing and displaying a web page having an embedded menu," duly and properly issued by the U.S. Patent and Trademark Office on April 21, 1998. A copy of the '768 patent is attached as Exhibit B.

18. The Defendant has been and/or is directly infringing and/or inducing others to infringe the '768 patent by, among other things, making, using, making available for another's use, offering to license or licensing in the United States, offering to sell or selling in the United States, products and/or services, including various web applications and services and the

Microsoft Corporation
1 Microsoft Way
Redmond, Washington 98052
Telephone: 425-882-8080

hardware and software running these applications and services, that embody or incorporate, or the operation of which otherwise practices, one or more claims of the '768 patent.

COUNT III

INFRINGEMENT OF U.S. PATENT NO. 5,644,737

19. Microsoft realleges and incorporates by reference the allegations set forth in paragraphs 1-12.

20. Microsoft is the owner of all right, title, and interest in U.S. Patent No. 5,644,737 ("the '737 patent"), entitled "Method and system for stacking toolbars in a computer display," duly and properly issued by the U.S. Patent and Trademark Office on July 1, 1997. A copy of the '737 patent is attached as Exhibit C.

21. The Defendant has been and/or is directly infringing and/or inducing others to infringe the '737 patent by, among other things, making, using, making available for another's use, offering to license or licensing in the United States, offering to sell or selling in the United States, or importing into the United States, products and/or services, including various web applications and services and the hardware and software running these applications and services, that embody or incorporate, or the operation of which otherwise practices, one or more claims of the '737 patent.

COUNT IV

INFRINGEMENT OF U.S. PATENT NO. 6,263,352

22. Microsoft realleges and incorporates by reference the allegations set forth in paragraphs 1-12.

23. Microsoft is the owner of all right, title, and interest in U.S. Patent No. 6,263,352 ("the '352 patent"), entitled "Automated web site creation using template driven generation of active server page applications," duly and properly issued by the U.S. Patent and Trademark Office on July 17, 2001. A copy of the '352 patent is attached as Exhibit D.

Microsoft Corporation
1 Microsoft Way
Redmond, Washington 98052
Telephone: 425-882-8080

24. The Defendant has been and/or is directly infringing the ’352 patent by, among other things, making, using, offering to license or licensing in the United States, offering to sell or selling in the United States, products and/or services, including various web applications and services and the hardware and software running these applications and services, that embody or incorporate, or the operation of which otherwise practices, one or more claims of the ’352 patent.

COUNT V

INFRINGEMENT OF U.S. PATENT NO. 6,122,558

25. Microsoft realleges and incorporates by reference the allegations set forth in paragraphs 1-12.

26. Microsoft is the owner of all right, title, and interest in U.S. Patent No. 6,122,558 (“the ’558 patent”), entitled “Aggregation of system settings into objects,” duly and properly issued by the U.S. Patent and Trademark Office on September 19, 2000. A copy of the ’558 patent is attached as Exhibit E.

27. The Defendant has been and/or is directly infringing and/or inducing others to infringe the ’558 patent by, among other things, making, using, making available for another’s use, offering to license or licensing in the United States, offering to sell or selling in the United States, products and/or services, including various web applications and services and the hardware and software running these applications and services, that embody or incorporate, or the operation of which otherwise practices, one or more claims of the ’558 patent.

COUNT VI

INFRINGEMENT OF U.S. PATENT NO. 6,542,164

28. Microsoft realleges and incorporates by reference the allegations set forth in paragraphs 1-12.

29. Microsoft is the owner of all right, title, and interest in U.S. Patent No. 6,542,164 (“the ’164 patent”), entitled “Timing and velocity control for displaying graphical information,”

Microsoft Corporation
1 Microsoft Way
Redmond, Washington 98052
Telephone: 425-882-8080

duly and properly issued by the U.S. Patent and Trademark Office on April 1, 2003. A copy of the '164 patent is attached as Exhibit F.

30. The Defendant has been and/or is directly infringing and/or inducing others to infringe the '164 patent by, among other things, making, using, offering to license or licensing in the United States, offering to sell or selling in the United States, products and/or services, including various web applications and services and the hardware and software running these applications and services, that embody or incorporate, or the operation of which otherwise practices, one or more claims of the '164 patent.

COUNT VII

INFRINGEMENT OF U.S. PATENT NO. 6,281,879

31. Microsoft realleges and incorporates by reference the allegations set forth in paragraphs 1-12.

32. Microsoft is the owner of all right, title, and interest in U.S. Patent No. 6,281,879 ("the '879 patent"), entitled "Timing and velocity control for displaying graphical information," duly and properly issued by the U.S. Patent and Trademark Office on August 28, 2001. A copy of the '879 patent is attached as Exhibit G.

33. The Defendant has been and/or is directly infringing and/or inducing others to infringe the '879 patent by, among other things, making, using, making available for another's use, offering to license or licensing in the United States, offering to sell or selling in the United States, products and/or services, including various web applications and services and the hardware and software running these applications and services, that embody or incorporate, or the operation of which otherwise practices, one or more claims of the '879 patent.

COUNT VIII

INFRINGEMENT OF U.S. PATENT NO. 5,845,077

34. Microsoft realleges and incorporates by reference the allegations set forth in paragraphs 1-12.

Microsoft Corporation
1 Microsoft Way
Redmond, Washington 98052
Telephone: 425-882-8080

35. Microsoft is the owner of all right, title, and interest in U.S. Patent No. 5,845,077 ("the '077 patent"), entitled "Method and system for identifying and obtaining computer software from a remote computer," duly and properly issued by the U.S. Patent and Trademark Office on December, 1, 1998. A copy of the '077 patent is attached as Exhibit H.

36. The Defendant has been and/or is directly infringing and/or inducing others to infringe the '077 patent by, among other things, making, using, making available for another's use, offering to license or licensing in the United States, offering to sell or selling in the United States, products and/or services, including various web applications and services and the hardware and software running these applications and services, that embody or incorporate, or the operation of which otherwise practices, one or more claims of the '077 patent.

COUNT IX

INFRINGEMENT OF U.S. PATENT NO. 5,941,947

37. Microsoft realleges and incorporates by reference the allegations set forth in paragraphs 1-12.

38. Microsoft is the owner of all right, title, and interest in U.S. Patent No. 5,941,947 ("the '947 patent"), entitled "System and method for controlling access to data entities in a computer network," duly and properly issued by the U.S. Patent and Trademark Office on August 24, 1999. A copy of the '947 patent is attached as Exhibit I.

39. The Defendant has been and/or is directly infringing the '947 patent by, among other things, making, using, making available for another's use, offering to license or licensing in the United States, offering to sell or selling in the United States, products and/or services, including various web applications and services and the hardware and software running these applications and services, that embody or incorporate, or the operation of which otherwise practices, one or more claims of the '947 patent.

Microsoft Corporation
1 Microsoft Way
Redmond, Washington 98052
Telephone: 425-882-8080

DEMAND FOR JURY TRIAL

40. Pursuant to Rule 38(b) of the Federal Rules of Civil Procedure, Microsoft respectfully requests a trial by jury on all issues properly triable by jury.

PRAYER FOR RELIEF

WHEREFORE, Microsoft prays for relief as follows:

A. For a judgment declaring that Defendant has infringed at least one claim of each of the Microsoft patents-in-suit;

B. For a judgment awarding Microsoft compensatory damages as a result of Defendant's infringement of the Microsoft patents-in-suit, together with interest and costs, and in no event less than a reasonable royalty;

C. For a judgment declaring that Defendant's infringement of the Microsoft patents-in-suit has been willful and deliberate;

D. For a judgment awarding Microsoft treble damages and pre-judgment interest under 35 U.S.C. § 284 as a result of Defendant's willful and deliberate infringement of the Microsoft patents-in-suit;

E. For a judgment declaring that this case is exceptional and awarding Microsoft its expenses, costs, and attorneys fees in accordance with 35 U.S.C. §§ 284 and 285 and Rule 54(d) of the Federal Rules of Civil Procedure;

F. For a grant of preliminary and permanent injunctions pursuant to 35 U.S.C. § 283, enjoining Defendant from further acts of infringement; and

G. For such other and further relief as the Court deems just and proper.

Dated: May 18, 2010

By: /s/ David E. Killough
David E. Killough

T. ANDREW CULBERT (SBN 35925)
andycu@microsoft.com
DAVID E. KILLOUGH (SBN 40185)
davkill@microsoft.com

Microsoft Corporation
1 Microsoft Way
Redmond, Washington 98052
Telephone: 425-882-8080

MICROSOFT CORPORATION
1 Microsoft Way
Redmond, Washington 98052
Telephone: 425-882-8080
Facsimile: 425-869-1327

OF COUNSEL:

DAVID T. PRITIKIN
dpritikin@sidley.com
RICHARD A. CEDEROTH
rcederoth@sidley.com
DOUGLAS I. LEWIS
dilewis@sidley.com
JOHN W. MCBRIDE
jwmcbride@sidley.com
SIDLEY AUSTIN LLP
One South Dearborn
Chicago, IL 60603
Telephone: 312-853-7000
Facsimile: 312-853-7036

Attorneys for Plaintiff Microsoft Corp.

Exhibit A

US007251653B2

(12) **United States Patent**
Huang et al.

(10) **Patent No.: US 7,251,653 B2**
(45) **Date of Patent: Jul. 31, 2007**

(54) **METHOD AND SYSTEM FOR MAPPING BETWEEN LOGICAL DATA AND PHYSICAL DATA**

(75) Inventors: **Chih-Jen Huang**, Kirkland, WA (US); **Steven Sheldon**, San Diego, CA (US); **Robert Turner**, Seattle, WA (US); **Patrick Conlan**, Redmond, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 497 days.

(21) Appl. No.: **10/880,888**

(22) Filed: **Jun. 30, 2004**

(65) **Prior Publication Data**

US 2006/0004750 A1 Jan. 5, 2006

(51) **Int. Cl.**
G06F 17/30 (2006.01)
G06F 7/00 (2006.01)

(52) **U.S. Cl.** **707/6**; 707/100; 707/203

(58) **Field of Classification Search** ..... 707/1–6, 707/100–102, 103 R, 104.1, 201, 203; 717/137, 717/141

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,734,887 A * 3/1998 Kingberg et al. 707/4
5,873,096 A * 2/1999 Lim et al. 707/201
5,937,402 A * 8/1999 Pandit 707/4
6,295,533 B2 * 9/2001 Cohen 707/5
6,457,003 B1 * 9/2002 Gajda et al. 707/4
6,490,590 B1 * 12/2002 Fink 707/100
6,687,704 B1 * 2/2004 Russell 707/100
6,711,582 B2 * 3/2004 Aldridge et al. ..... 707/103 Y
7,062,502 B1 * 6/2006 Kesler 707/102

OTHER PUBLICATIONS

Tolkin, Steven, "Aggregation Everywhere: Data Reduction and Transformation in the Phoenix Data Warehouse," Nov. 1999 (8 pages).
Michalk, Dale, "The DataSet Object: At Your Web Service," Copyright 2001-2003, Fawcette Technical Publications, (7 pages), http://www.fawcette.com/xmlmag/2001_1_11/magazine/columns/integration/dmichalk/default)pf.aspx.
NET Framework Class Library, DataSet Class, Copyright 2004 Microsoft Corporation (4 pages) http://msdn.microsoft.com/library/en-us/cpref/html/frlrfSystemDataDataSetClassTopic.asp?frame=true.

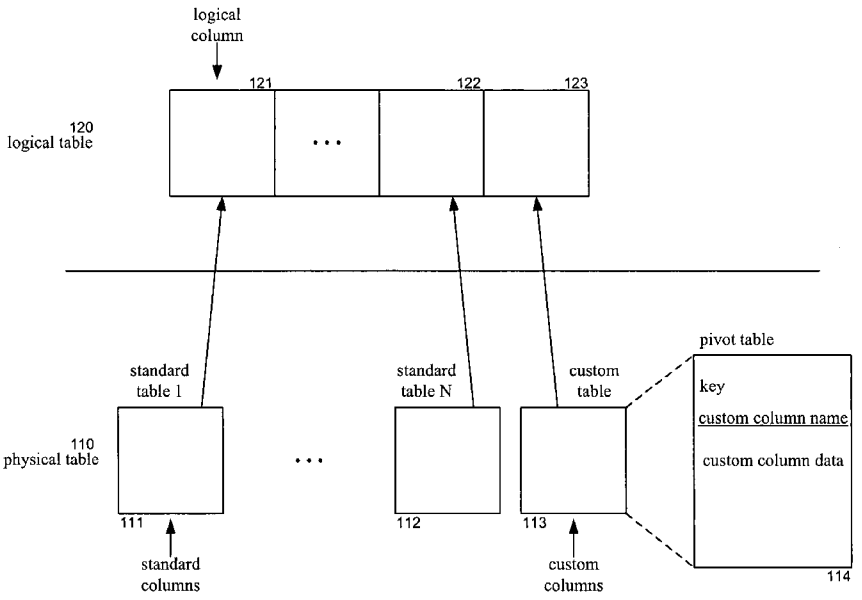
* cited by examiner

Primary Examiner—Jeffrey Gaffin
Assistant Examiner—Jacques Veillard
(74) *Attorney, Agent, or Firm*—Perkins Coie LLP

(57) **ABSTRACT**

The mapping system maps a physical table of a database to a logical table representing a logical view of the database that integrates standard columns and custom columns. The physical table includes a standard table with standard columns and a custom table with custom columns. The custom table may be implemented as a pivot table. The mapping system provides a map between standard and custom columns and logical columns. The physical table may include multiple standard tables. The mapping system allows for individual standard tables to be updated, rather than updating all the columns across all the standard tables for a row.

27 Claims, 9 Drawing Sheets



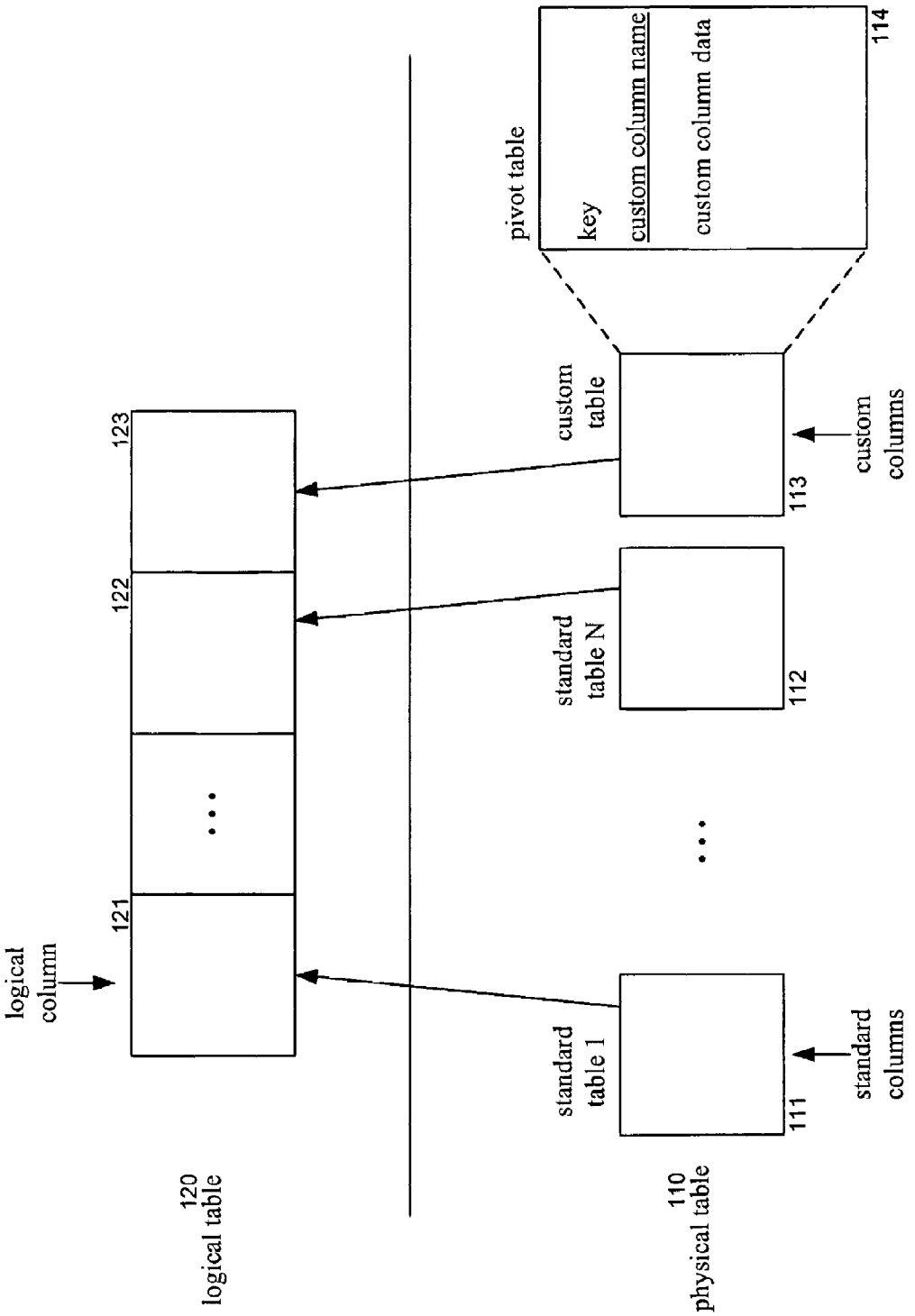

pivot table
key
custom column name
custom column data
114
custom table
113
custom columns
standard table N
112
standard table 1
111
standard columns
logical column
121
122
123
120 logical table
110 physical table

FIG. 1

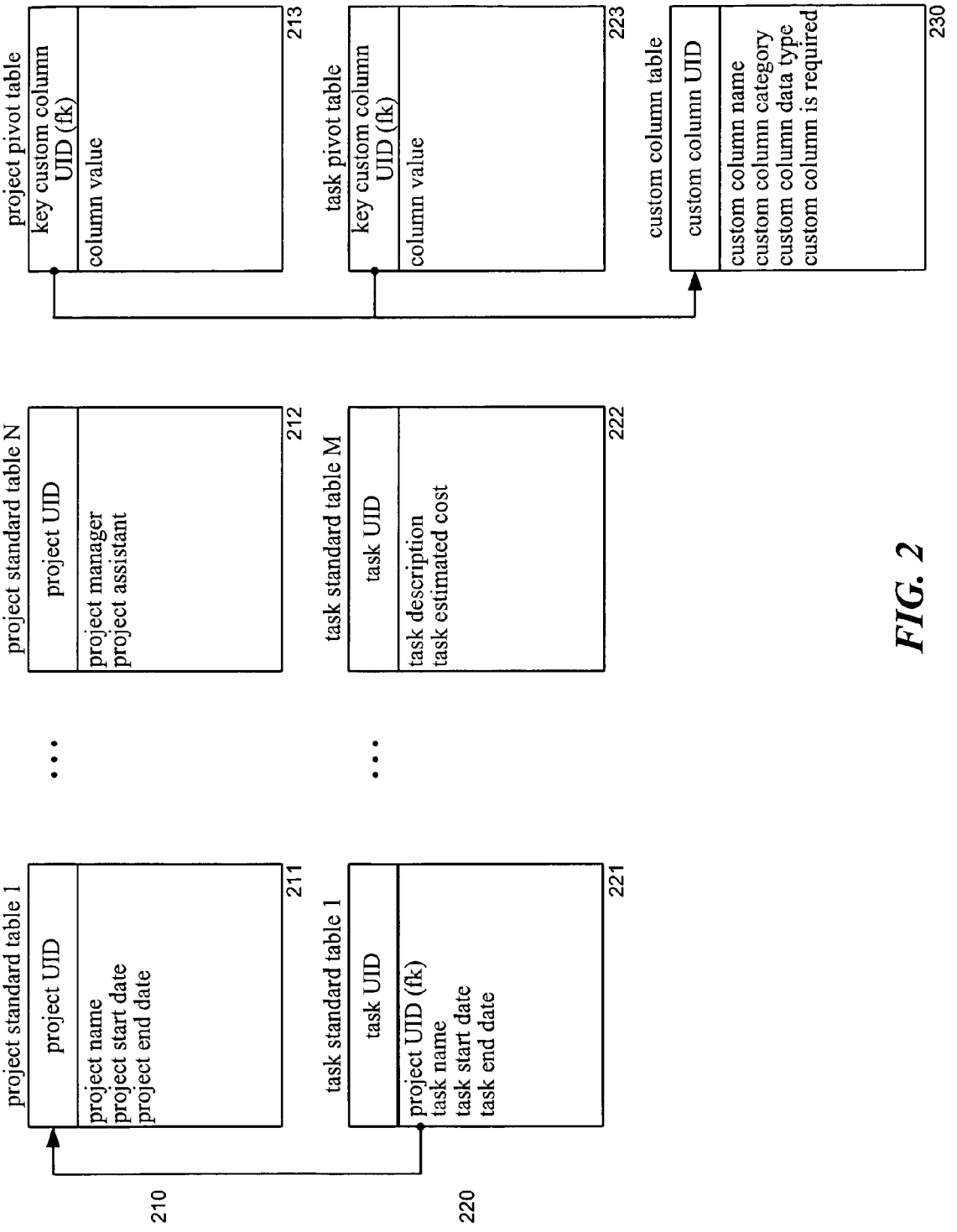

project standard table 1
project UID
project name
project start date
project end date
211
project standard table N
project UID
project manager
project assistant
212
task standard table 1
task UID
project UID (fk)
task name
task start date
task end date
221
task standard table M
task UID
task description
task estimated cost
222
project pivot table
key custom column UID (fk)
column value
213
task pivot table
key custom column UID (fk)
column value
223
custom column table
custom column UID
custom column name
custom column category
custom column data type
custom column is required
230
210
220

FIG. 2

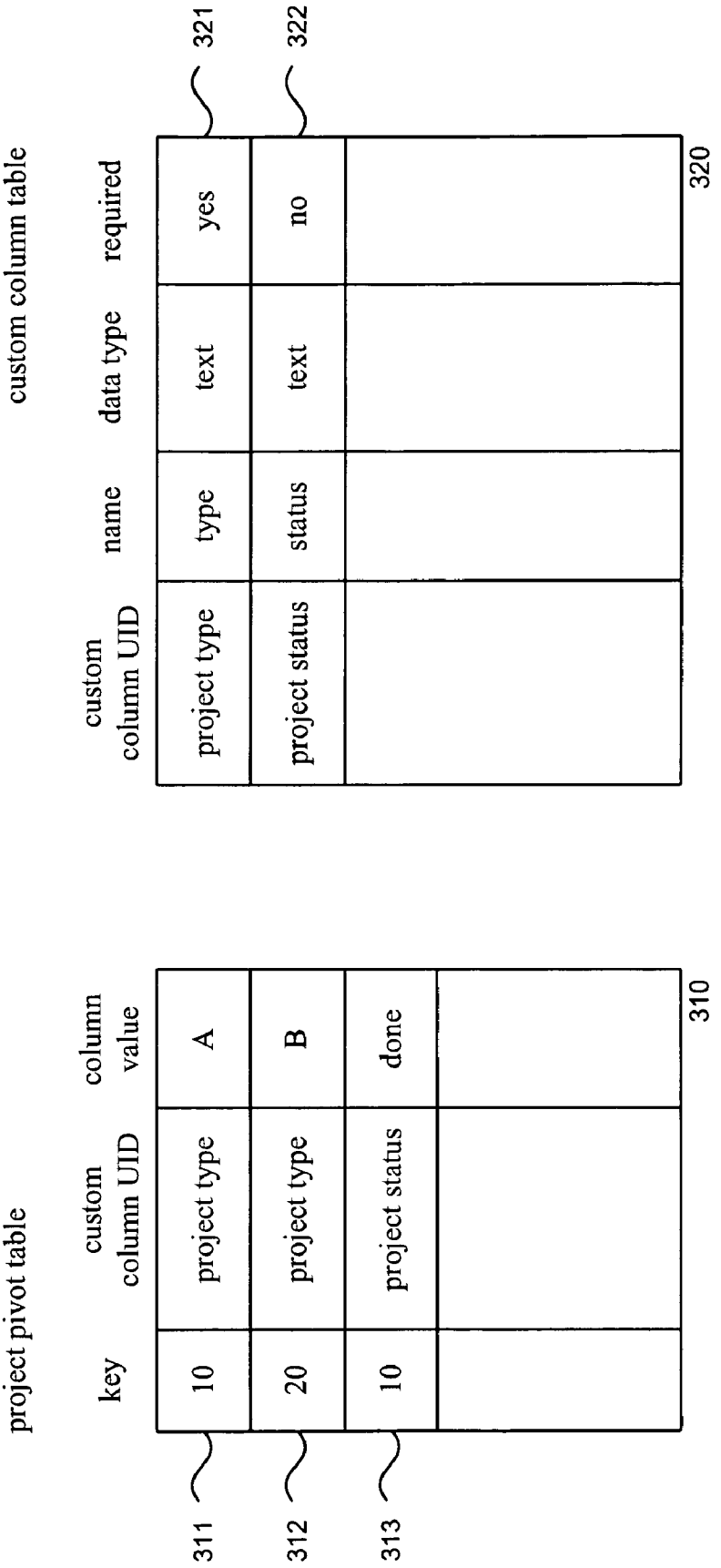

custom column table
custom column UID
name
data type
required
project type
type
text
yes
project status
status
text
no
321
322
320
project pivot table
key
custom column UID
column value
10
project type
A
20
project type
B
10
project status
done
310
311
312
313

FIG. 3

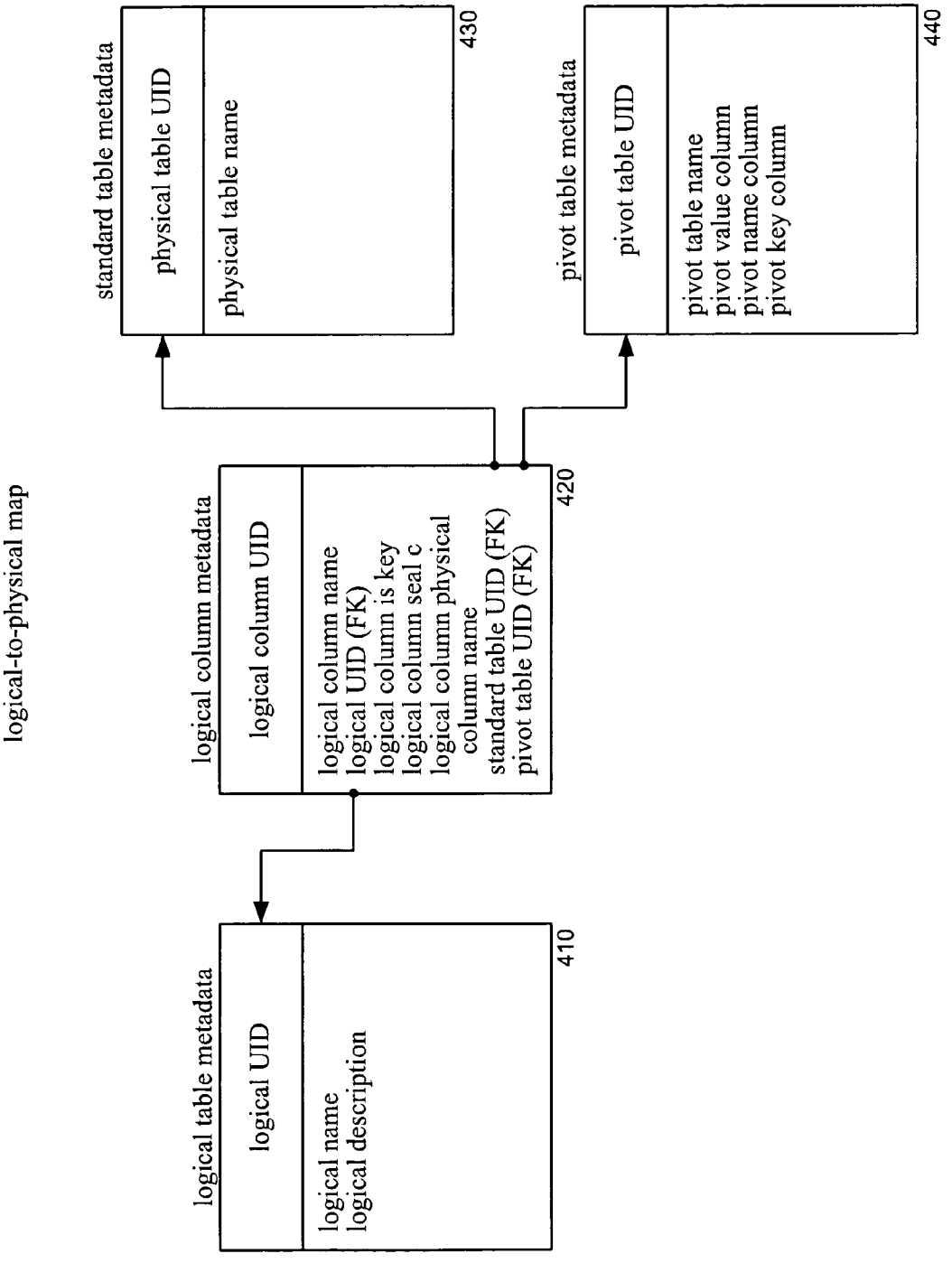

logical-to-physical map
logical table metadata
logical UID
logical name
logical description
410
logical column metadata
logical column UID
logical column name
logical UID (FK)
logical column is key
logical column seal c
logical column physical column name
standard table UID (FK)
pivot table UID (FK)
420
standard table metadata
physical table UID
physical table name
430
pivot table metadata
pivot table UID
pivot table name
pivot value column
pivot name column
pivot key column
440

FIG. 4

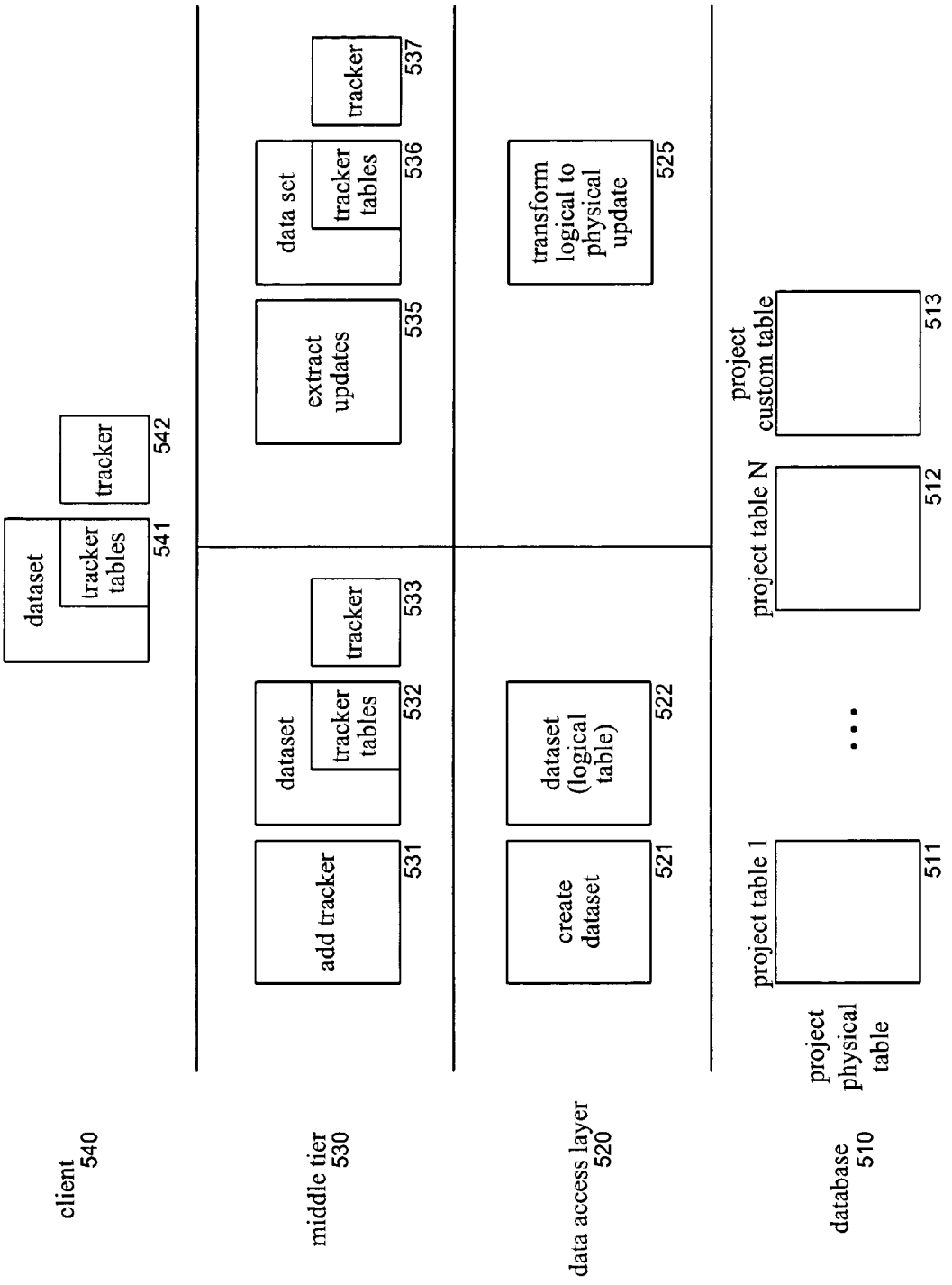

client 540
middle tier 530
data access layer 520
database 510
project physical table
dataset
tracker tables
541
tracker
542
add tracker
531
dataset
tracker tables
532
tracker
533
create dataset
521
dataset (logical table)
522
extract updates
535
data set
tracker tables
536
tracker
537
transform logical to physical update
525
project table 1
511
project table N
512
project custom table
513

FIG. 5

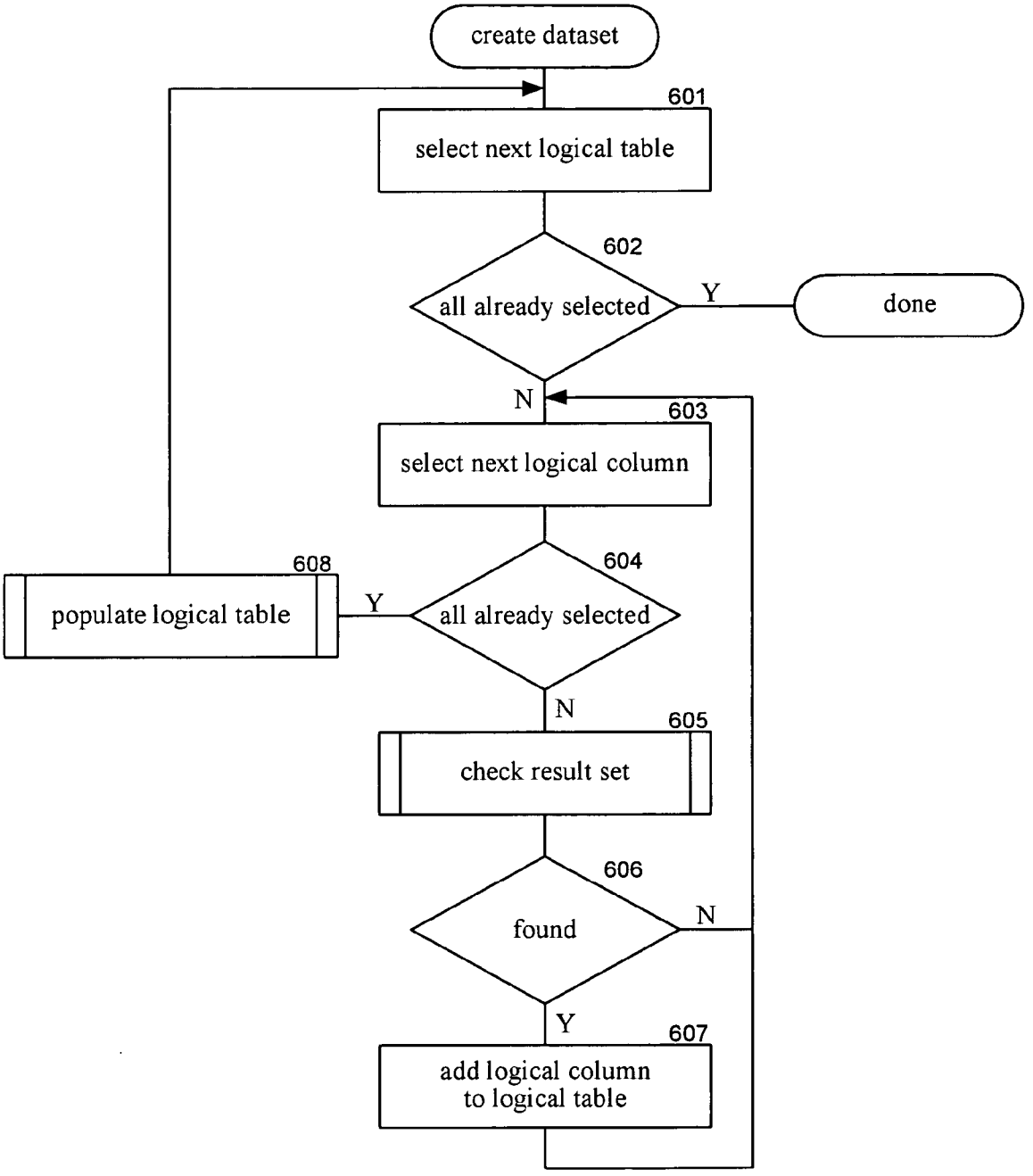

create dataset
601
select next logical table
602
all already selected
Y
done
N
603
select next logical column
608
populate logical table
Y
604
all already selected
N
605
check result set
606
found
N
Y
607
add logical column to logical table

FIG. 6

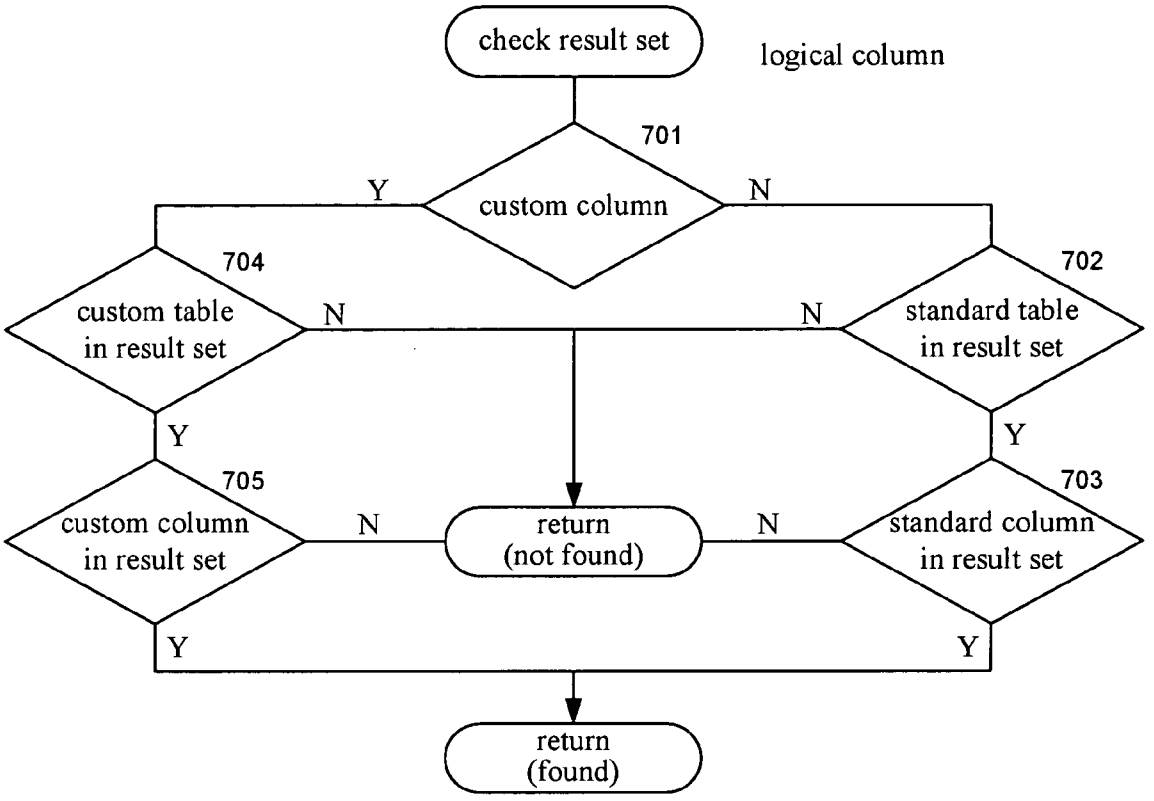

check result set
logical column
701
custom column
Y
N
704
custom table in result set
N
N
702
standard table in result set
Y
Y
705
custom column in result set
N
return (not found)
N
703
standard column in result set
Y
Y
return (found)

FIG. 7

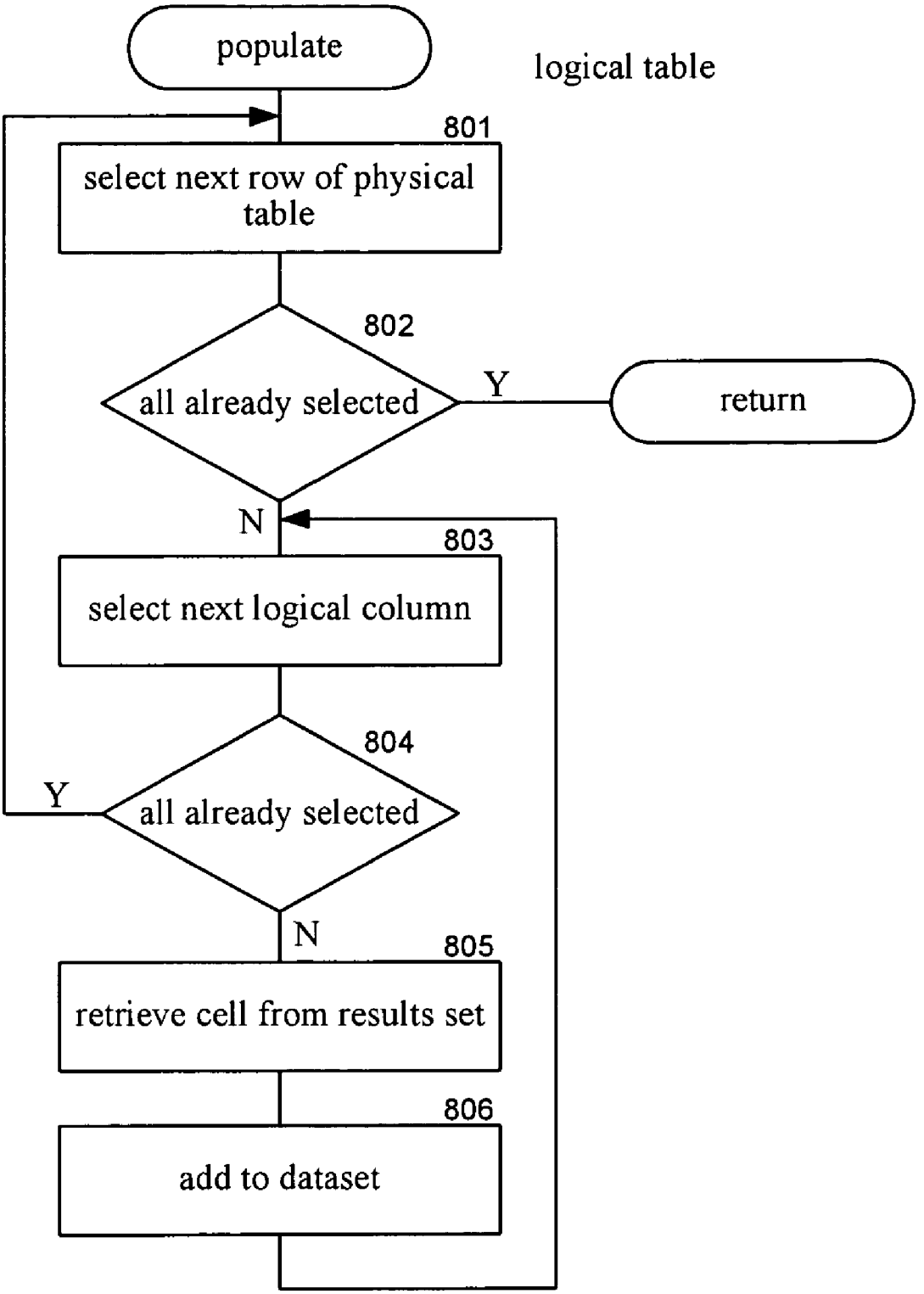

populate
logical table
801
select next row of physical table
802
all already selected
Y
return
N
803
select next logical column
804
all already selected
Y
N
805
retrieve cell from results set
806
add to dataset

FIG. 8

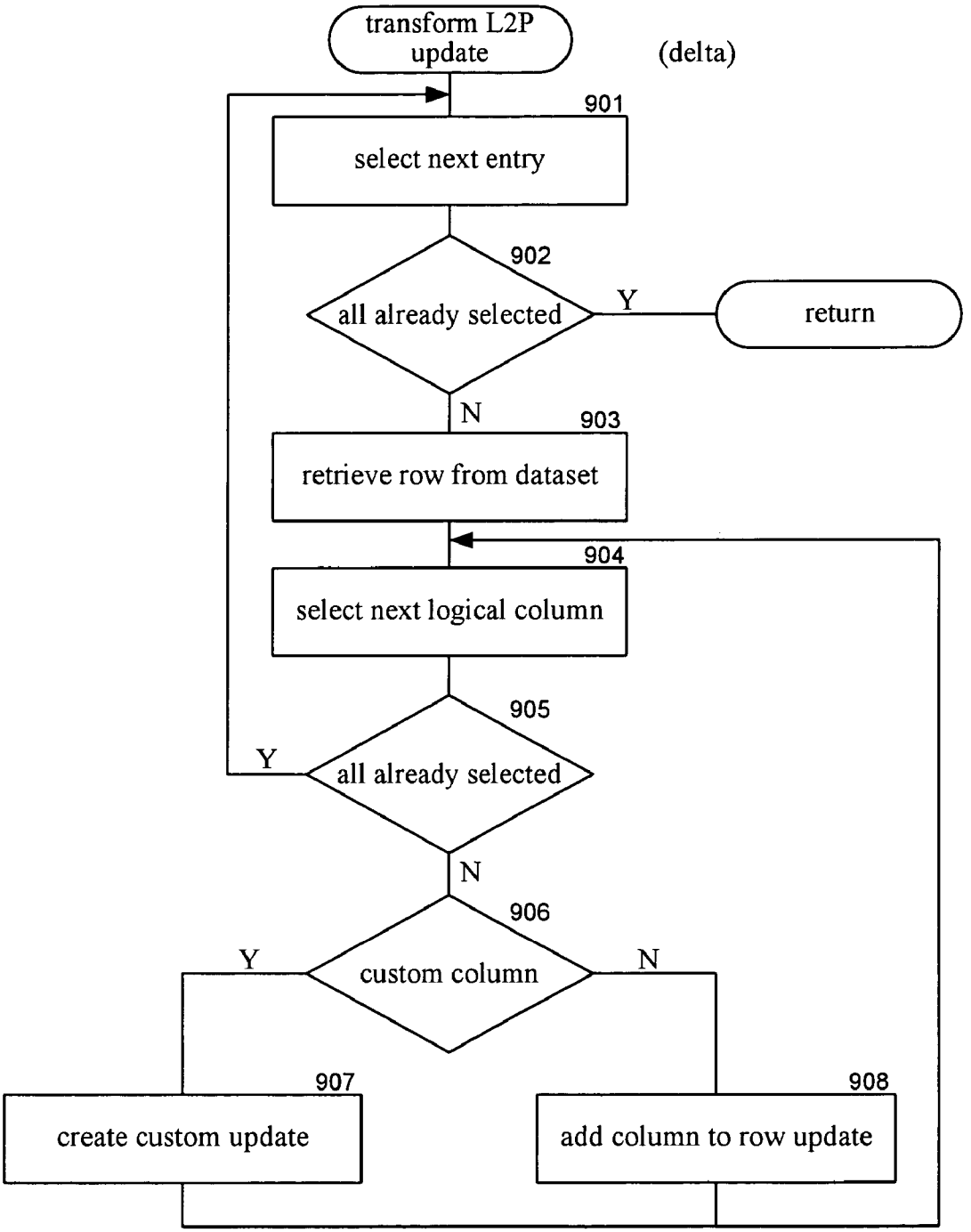

transform L2P update
(delta)
901
select next entry
902
all already selected
Y
return
N
903
retrieve row from dataset
904
select next logical column
905
all already selected
Y
N
906
custom column
Y
N
907
create custom update
908
add column to row update

FIG. 9

METHOD AND SYSTEM FOR MAPPING BETWEEN LOGICAL DATA AND PHYSICAL DATA

TECHNICAL FIELD

The described technology relates generally to mapping between logical data and physical data and including to mapping when the physical data includes custom data.

BACKGROUND

Many applications use a database to store their data. The database for an application is typically designed by the developer of the application to include a table for each entity used by the application. Each entity table contains a row for each specific entity and various columns for storing properties of the entity. For example, in the case of a project management application, the entities may include a project, a task, an assignment, or a resource, and a specific entity is a specific project, a specific task, a specific assignment, or a specific resource. The project table may contain a project identifier column, a project name column, a project start date column, and so on. The project identifier column contains the unique identifier of a specific project and is referred to as a "unique key" of the project table. Each row of the project table corresponds to a specific project, and the cells of a row contain the data of that specific project for the columns. A task table may contain a task identifier column, project identifier column, task name column, and so on. The task identifier column contains the unique identifiers of specific tasks. The project identifier column contains the project identifier of the specific project with which the task is associated and is referred to as a "foreign key." Each row of the task table corresponds to a specific task.

Complex applications may have many hundreds of properties associated with an entity. This presents problems for databases that limit the number of columns of a table. For example, some databases may limit the number of columns to 128 or 256. To overcome this problem, applications may store data for an entity in multiple database tables. For example, if an application needs 300 columns to represent the properties of an entity and the limit on the number of columns of a table is 128, then the developer of the application may divide the 300 columns across three tables with 101 columns in each table. Each table may contain a unique key column and 100 property columns. When the properties of a specific entity is added to the database, the application generates a unique identifier for that specific entity and adds a row to each of the three tables with its unique key set to that unique identifier. The combination of the rows from the three tables with the same unique identifier corresponds to the columns for the entity. To access the data for that specific entity, the application may join the three tables. As a result, at least for viewing purposes, the join results in a logical data view that contains the unique identifier column and the 300 property columns.

Even though these complex applications have many properties associated with an entity, referred to as "standard" properties or columns, users may need to have additional properties associated with an entity. For example, in the case of a project management application, a user may need to track project type and project status, which may have no corresponding standard column. To assist users in defining their own properties for an entity, applications may allow custom columns to be defined. For example, a user may define a type custom column and a status custom column to track the type and status of projects. The custom column can be considered just one more column associated with an entity.

Although custom columns could be supported by modifying the schema of the database, such modifications can be time-consuming and error-prone, especially if performed by the users of the application. To allow users the flexibility to create custom columns without modifying the schema of the database, some applications use a "pivot" table to store information relating to custom columns. A pivot table for an entity would typically include a key column, a custom column name column, and a data column. Whenever data for a custom column is to be added for a specific entity, a new row is added to the pivot table that contains the unique key associated with that specific entity, the name of the custom column, and the data.

The use of pivot tables to represent custom columns may make it difficult for a user to retrieve all the properties associated with a specific entity. In particular, although a join can be used to combine the data of standard tables, the data of the custom columns cannot be joined so easily. Moreover, even if with only standard tables are joined to provide a logical data view, some databases may not allow updates via the logical data view. It would be desirable to provide a logical data view that would integrate both standard columns and custom columns and would allow for the updating of data of both standard columns and custom columns via a logical data view.

SUMMARY

A method and system for providing a logical view of data that combines standard and custom fields is provided. The system creates a logical view of physical data that includes standard data of standard fields and custom data of custom fields. The system has a map that maps logical fields of logical data to the corresponding standard fields or custom fields of the physical data. The system uses the map to generate the logical view. When the custom fields are represented by pivot data, the system converts the pivot data so that it appears as a logical field. The system may allow the updating of data of a custom field via the logical view and a standard field when the standard fields are represented as standard columns of multiple standard database tables.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram that illustrates a physical table of a database representing an entity and a corresponding logical table in one embodiment.

FIG. **2** is a block diagram that illustrates physical tables for a project management application in one embodiment.

FIG. **3** is a block diagram that illustrates sample data of a project pivot table and custom column table in one embodiment.

FIG. **4** is a block diagram that illustrates a schema for a map of logical data to physical data in one embodiment.

FIG. **5** is a block diagram illustrating the interaction of components of the mapping system in one embodiment.

FIG. **6** is a flow diagram that illustrates the create dataset object component in one embodiment.

FIG. **7** is a flow diagram that illustrates the processing of the check result set component in one embodiment.

FIG. **8** is a flow diagram that illustrates the processing of the populate logical table component in one embodiment.

FIG. **9** is a flow diagram that illustrates the processing of the transform logical to physical update component in one embodiment.

DETAILED DESCRIPTION

A method and system for providing a view of data that combines standard and custom data is provided. In one embodiment, a mapping system provides a map between physical fields of physical data and logical fields of logical data. The physical fields may include standard fields and custom fields. The custom fields may be represented using pivot data. To create a view of the physical data, the physical data is queried to generate a result set that includes custom fields represented using pivot data and standard fields. The mapping system uses the map to generate a logical data view that integrates standard and custom fields in a way that hides from a user or client the distinction between standard and custom field. In addition, the mapping system tracks updates to the logical data and then updates the corresponding physical data. The mapping system may keep a log of the updates that are made to the logical data. The mapping system uses the map to identify which standard fields and custom fields need to be updated and updates them accordingly. In this way, the distinction between standard fields and custom fields is hidden from the logical data view and updates made to the logical data view can be reflected in the physical data.

In one embodiment, the mapping system maps a physical table of a database to a logical table representing a logical view that integrates standard columns and custom columns. The physical table includes a standard table with standard columns and a custom table with custom columns. The custom table may be implemented as a pivot table. The mapping system provides a map between standard and custom columns and logical columns. The map may include for each logical column of the logical table an indication of the corresponding standard column and standard table or an indication of the corresponding custom column. The pivot table may include a key column, custom column name column, and data column. The set of unique custom column names within the custom column name column of the pivot table represents all the custom columns that have been defined for the physical table. In one embodiment, the name of the pivot table and its column names may be hard-coded into the mapping system. Alternatively, the map may map each logical column that corresponds to a custom column to the name of the corresponding pivot table and the names of the columns within the pivot table corresponding to the key, custom column name, and data columns. The mapping system may represent a logical table as a dataset object that defines a logical view and methods for accessing the logical data. (See, D. Michalk, "The DataSet Object: At Your Web Service," XML & Web Services Magazine, October/November 2001, which is hereby incorporated by reference.) The mapping system may add functionality to the dataset object to track changes that are made to the data within the dataset object. When the changes made to the logical table are to be committed to the physical table, the mapping system processes each change by mapping the updated columns of the logical table to the corresponding physical columns of the physical table. The updated columns may correspond to standard columns or custom columns. If an updated column corresponds to a custom column, then the mapping system updates the corresponding pivot table as appropriate.

In one embodiment, the physical table may include multiple standard tables, for example, if the database limits the number of columns within a table to less than the number needed to represent all the properties of an entity. The mapping system allows for individual standard tables to be updated, rather than updating all the columns across all the standard tables for a row. Prior techniques for updating a view that included a join of multiple tables may have required that all the columns of all the tables be updated even when only a single column of the view is updated. The mapping system may also define a logical table to contain logical columns corresponding to different physical tables. For example, a logical table may contain a row for each task with logical columns corresponding to various physical columns of the task physical table and a physical column for the project physical table.

FIG. **1** is a block diagram that illustrates a physical table of a database representing an entity and a corresponding logical table in one embodiment. The physical table **110** includes standard tables **111-112** and a custom table **113**. Each standard table includes a unique key standard column and other standard columns that each correspond to a property of the entity represented by the physical table. The custom table is implemented as a pivot table **114**. The custom table, however, logically includes a unique key column and each custom column. A row of the physical table for a specific entity, identified by a unique identifier, corresponds to a join of the standard tables and the custom table. The mapping system generates the logical table **120**, which may be represented as a dataset object, corresponding to the physical table by creating a logical join of the standard tables and the custom table. The join with the custom table is logical in the sense that the custom table is a logical representation of the pivot table. The mapping system converts the rows of the pivot table to the corresponding column of the custom table to effect the logical join.

FIG. **2** is a block diagram that illustrates physical tables for a project management application in one embodiment. The physical tables include a project table **210** corresponding to a project entity and a task table **220** corresponding to a task entity. The project table includes project standard tables **211-212** and project pivot table **213**. The task table includes task standard table **221-222** and task pivot table **223**. The project standard tables contain a unique project identifier column and various standard columns relating to project properties. The project pivot table contains an entry for each cell of the project table that contains a custom value. The project pivot table includes a key column that contains the project unique identifier, the custom column unique identifier column, and a data column. The custom column unique identifier column is a reference to a row in a custom column table **230**. The custom column table contains a row for each custom column that has been defined. Each row contains name, category, and data type of a custom column. The task standard tables contain a task unique identifier column and various standard columns relating to task properties. The task pivot table contains an entry for each cell of the task table in a manner similar to the project pivot table.

FIG. **3** is a block diagram that illustrates sample data of a project custom table represented as a project pivot table and custom column table in one embodiment. The custom column table **320** includes rows **321-322**. Row **321** defines the custom column "project" with a data type of "text" and an indication that the column is required to have a data value. Row **322** defines the custom column "status" with the data type of "text" and an indication that the column is not required to have a data value. The project pivot table **310** includes rows **311-313**. Row **311** corresponds to the cell for the "type" custom column for project **10**. This row indicates

that project **10** has a type of "A." Row **312** corresponds to the cell for the "type" custom column for project **20**. This row indicates that project **20** has a type of "B." Row **313** corresponds to the cell for the "status" custom column for project **10**. This row indicates that project **10** has a status of "done." The custom column unique identifier column of the project pivot table contains a foreign key to the custom column table. The custom column table thus contains a row for each custom column describing its characteristics.

FIG. **4** is a block diagram that illustrates a schema for a map of logical data to physical data in one embodiment. The schema defines logical table metadata **410**, logical column metadata **420**, standard table metadata **430**, and pivot table metadata **440**. The logical table metadata contains a row for each logical table corresponding to a physical table. In one embodiment, a logical table may be generated from multiple physical tables. The logical table metadata contains logical unique identifier, name, and description columns. The logical column metadata contains a row for each logical column. It includes a logical column unique identifier, name, logical table unique identifier (as a foreign key), is key, is calculated, physical column name, standard table unique identifier (as a foreign key), and pivot table unique identifier (as a foreign key) columns. The logical table unique identifier column maps the logical column to the corresponding logical table in which it is contained. The standard table unique identifier column maps the logical column to the corresponding standard table. The pivot table unique identifier maps the logical column to the corresponding row of the pivot table. The physical column name contains the column name associated with either the standard table or the pivot table. The standard table metadata has one row for each standard table and includes a physical table unique identifier and name column. The name identifies the name of the physical table. The pivot table metadata includes a row for each custom column and contains a pivot table unique identifier, name, value column, name column, and key column columns. The table name column specifies the name of the pivot table. The value column identifies the name of the column of the pivot table that contains the data value. The name column identifies the column of the pivot table that contains the name of the custom column. The key column identifies the column of the pivot that contains the key of the corresponding physical table.

FIG. **5** is a block diagram illustrating the interaction of components of the mapping system in one embodiment. The figure illustrates a database layer **510**, a data access layer **520**, a middle tier **530**, and a client **540**. The database may be connected to the data access layer via a network, and the client may be connected to the middle tier via a network. The database includes a project physical table comprising project standard tables **511**-**512** and a project custom table **513**. When a result set is generated for the project physical table, it is passed to the create dataset component **521** of the data access layer. If a network connects the database and the data access layer, then data sent via the network would typically need to be serialized and de-serialized. The create dataset component uses the logical-to-physical map to create a dataset object **522** that represents the logical table. The dataset object is passed to the add tracker component **531** of the middle tier. The add tracker component adds tracker tables to the dataset object **532** and adds tracker object **533**. The tracker object is responsible for tracking each update to the logical table of the dataset objects and storing an indication of the update in the tracker tables. The dataset object is then serialized (when the middle tier and client are connected via a network) and provided to the client. The client de-serializes the dataset object and instantiates dataset object **541** and tracker object **542**. The client accesses the dataset object to view and update to the logical tables of the dataset object. The tracker object logs all updates, such as updating a cell, adding a row, or deleting a row. Upon completion, the client serializes the dataset object and provides it to the middle tier. The middle tier de-serializes the dataset object and instantiates a dataset object **536** and tracker object **537**. The extract updates component extracts the update information from the tracker tables and provides that information to the transform logical to physical update component **525** of the data access layer. The transform logical to physical update component uses the logical-to-physical map to generate updates to the project physical table corresponding to the updates made by the client to the logical table. The component may generate a series of SQL statements.

The computing device on which the mapping system is implemented may include a central processing unit, memory, input devices (e.g., keyboard and pointing devices), output devices (e.g., display devices), and storage devices (e.g., disk drives). The memory and storage devices are computer-readable media that may contain instructions that implement the mapping system. In addition, the data structures and message structures may be stored or transmitted via a data transmission medium, such as a signal on a communications link. Various communications links may be used, such as the Internet, a local area network, a wide area network, or a point-to-point dial-up connection.

FIG. **5** illustrates an example of a suitable operating environment in which the mapping system may be implemented. The operating environment is only one example of a suitable operating environment and is not intended to suggest any limitation as to the scope of use or functionality of the mapping system. Other well-known computing systems, environments, and configurations that may be suitable for use include personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, programmable consumer electronics, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

The mapping system may be described in the general context of computer-executable instructions, such as program modules, executed by one or more computers or other devices. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Typically, the functionality of the program modules may be combined or distributed as desired in various embodiments.

FIG. **6** is a flow diagram that illustrates the create dataset object component in one embodiment. The component is passed a result set and creates a dataset object representing a logical view of the result set. In this embodiment, the component selects each logical table and logical column of the logical table and adds a column to that logical table if the corresponding physical column is represented in the result set. One skilled in the art will appreciate that selecting each of the physical columns of the result set can alternatively identify the logical columns for the logical tables. In block **601**, the component selects the next logical table that is defined in the logical-to-physical map. In decision block **602**, if all the logical tables have already been selected, then the component completes, else the component continues at block **603**. In block **603**, the component selects the next logical column of the selected logical table. In decision

block **604**, if all the logical columns of the selected logical table have already been selected, then the component continues at block **608**, else the component continues at block **605**. In block **605**, the component invokes the check result set component to determine whether the physical column corresponding to the selected logical column is in the result set. In decision block **606**, if the physical column is in the result set, then the component continues at block **607**, else the component loops to block **603** to select the next logical column. In block **607**, the component adds the selected logical column to logical table and then loops to block **603** to select the next logical column. In block **608**, the component invokes the populate logical table component to add rows to the logical table that are generated from the result set and loops to block **601** to select the next logical table.

FIG. **7** is a flow diagram that illustrates the processing of the check result set component in one embodiment. This component is passed an indication of a logical column and returns an indication as to whether the corresponding physical column is in the result set. In decision block **701**, if the logical column corresponds to a custom column, then the component continues at block **704**, else the component continues at block **702**. In decision block **702**, if the standard table that contains the standard column corresponding to the logical column is in the result set, then the component continues at block **703**, else the component returns an indication of not found. In decision block **703**, if the standard column corresponding to the logical column is in the result set, then the component returns an indication of found, else the component returns an indication of not found. In decision block **704**, if the custom table corresponding to logical column is in the result set, then the component continues at block **705**, else the component returns an indication of not found. In decision block **705**, if the custom column corresponding to logical column is in the result set, then the component returns an indication of found, else the component returns an indication of not found.

FIG. **8** is a flow diagram that illustrates the processing of the populate logical table component in one embodiment. The component is passed an indication of a logical table of the dataset object and adds rows to the logical table corresponding to the data of the result set. In block **801**, the component selects the next row of the physical table corresponding to the logical table. In decision block **802**, if all the rows of the physical table have already been selected, then the component returns, else the component continues at block **803**. In block **803**, the component selects the next logical column of the logical table. In decision block **804**, if all the logical columns of the logical table have already been selected, then the component loops to block **801** to select the next row of the physical table, else the component continues at block **805**. In block **805**, the component retrieves the data of the cell from the result set for the selected logical column of the selected row of the physical table. In block **806**, the component adds retrieved data to logical column of the logical table and then loops to block **803** to select the next logical column.

FIG. **9** is a flow diagram that illustrates the processing of the transform logical to physical update component in one embodiment. The component is passed a delta data structure that defines various updates to the logical table. The delta data structure contains an entry for each update of a logical table. Each entry identifies a logical table that was updated, logical columns of the logical table that were updated, an operation (e.g., update, delete, or add), and the name of a key and its value which specify the specific row of the logical table that was updated. The entry also contains for each logical column indications of whether to generate a new identifier for this column, whether the data was null before the update, and whether the data is null after the update. In block **901**, the component selects the next entry specified in the delta data structure. In decision block **902**, if all the entries of the delta data structure have already been selected, then the component returns, else the component continues at block **903**. In block **903**, the component retrieves the row from the dataset object corresponding to the updated row of the logical table of the selected entry. In block **904**, the component selects the next logical column specified in the delta data structure for the selected entry. In block **905**, if all the logical columns have already been selected, then the component loops to block **901** to select the next entry of the delta data structure, else the component continues at block **906**. In decision block **906**, if the selected logical column is a custom column, then the component continues at block **907**, else the component continues at block **908**. In block **907**, the component generates update instructions (e.g., SQL statements) for the database to update the pivot table for the custom column corresponding to the selected logical column and then loops to block **904** to select the next logical column. In block **908**, the component adds a column to a row update instruction for the standard table that contains the logical column. The component then loops to block **904** to select the next logical column.

The Pseudo Code Table contains sample pseudo code for generating the instructions to update the physical table based on the delta data structure.

Pseudo Code Table

```
1.  GenerateSqlFromDelta( )
2.  {
3.  For each entry in delta
4.      Read operation
5.      Read all keys into collection accessible by name
6.      Read all columns into collection accessible by name
7.
8.      For each standard or custom table of the logical table being updated
9.          If table is a custom table
10.             For each column in the custom table
11.                 Call GeneratePivotUpdate(operation, keys, update)
12.             Next
13.         Else
14.             For each column in the standard table
15.                 Add column name and update value to update list
16.             Next
```

-continued

Pseudo Code Table

```
17.
18.                 Call GenerateUpdate(operation, keys, update list)
19.             Endif
20.         Next
21.     Next
22.     }
23.
24.     GeneratePivotUpdate(operation, keys, update)
25.     {
26.         Lookup pivot table definition (definition defines key columns + name
        column)
27.
28.         If the operation is an update, generate an insert if the value is currently null,
        an update if it is going from a value to another value, and generate a delete if it
        changes from a value to null.
29.
30.         Inserts and deletes generate insert and delete statements respectively.
31.     }
32.
33      GenerateUpdate(operation, keys, updatelist)
34.     {
35.         Generate where clause from keys
36.
37.         If operation is delete
38.             Generate delete statement only using keys.
39.         Else
40.             If operation is insert
41.                 Generate insert using keys and update list as values.
42.             Else
43.         Generate update using update list and use where clause from earlier in
            function.
44.
45.             Endif
46.         Endif
47.     }
```

One skilled in the art will appreciate that although specific embodiments of the mapping system have been described herein for purposes of illustration, various modifications may be made without deviating from the spirit and scope of the invention. One skilled in the art will appreciate that a pivot table can be organized in many different ways. For example, multiple entities can share a common pivot table or each entity can have its own pivot table. Also, a pivot table can be represented as a single database table or multiple database tables. A pivot table contains data for the custom columns of a physical table without having a database column for each custom column. Accordingly, the invention is not limited except by the appended claims.

We claim:

1. A method in a computer system for providing a view of data, the method comprising:

providing physical data having standard and custom data, the standard data having entries with data for standard fields, the custom data having data for custom fields, the custom fields being represented by pivot data;

providing a map between standard and custom fields and logical fields of logical data;

providing a result set containing physical data from a standard field and a custom field;

organizing the physical data of the result set into logical data using the provided map; and

storing the organized logical data as the view.

2. The method of claim **1** wherein the physical data, the custom data, the pivot data, and the logical data are represented as tables, the tables having rows and columns.

3. The method of claim **2** where each row of the pivot table identifies a custom column, a row of the physical table, and data for the custom column of the identified row of the physical table.

4. The method of claim **2** wherein the map maps columns of the logical table to the corresponding standard column or custom column.

5. The method of claim **4** wherein the map of a logical column includes an identifier of the custom column used by the pivot data.

6. The method of claim **2** wherein when the logical table is updated, updating the standard table and the custom table.

7. The method of claim **2** wherein the physical table comprises multiple standard tables with standard columns and when a logical column of the logical table is updated, updating only the standard table including the corresponding standard column.

8. The method of claim **2** wherein when the updating of the logical table includes adding data for a custom column of a logical row, adding a row to the pivot table for the custom column of the physical row corresponding to the logical row.

9. The method of claim **2** wherein when the updating of the logical table includes updating data for a custom column of a logical row, updating a row of the pivot table for the custom column of the physical row corresponding to the logical row.

10. A computer-readable storage medium containing a data structure for mapping between a logical table and a physical table, the physical table including a standard table and a custom table, the data structure comprising:

for each logical column of the logical table,

when the logical column corresponds to a standard column of the standard table, mapping the logical column to the corresponding standard column; and

when the logical column corresponds to a custom column of the custom table, mapping the logical column to the corresponding custom column, the custom table being represented by a pivot table;

instructions for generating a logical view of physical data using the mapping; and

instructions for storing the generated view.

11. The computer-readable storage medium of claim **10** wherein the pivot table includes for each custom column, a row for each row of the physical table including data for the each custom column.

12. The computer-readable storage medium of claim **10** wherein each mapping of a logical column includes a name for the logical column, an indication of whether the corresponding standard column is a key of the standard table of the corresponding standard column, a name for the corresponding physical column, and a indication of either the standard table or the custom table.

13. The computer-readable storage medium of claim **10** wherein the mapping of the logical column to the corresponding custom column includes a name of the corresponding pivot table, an identifier of a pivot column containing a name of the custom column, an identifier of a pivot column containing data of the custom column, and an identifier of a pivot column containing a key for a physical row.

14. A computer-readable storage medium containing instructions for controlling a computer system to update data, by a method comprising:

providing a map between standard and custom columns of a physical table and logical columns of a logical table, the custom columns being represented using a pivot table;

providing an indication of an update to the logical table;

using the map to determine the standard column or custom column to which an updated logical column corresponds; and

effecting the update of the determined column of the physical table.

15. The computer-readable storage medium of claim **14** wherein when the determined column is a custom column updating a row of the pivot table.

16. The computer-readable storage medium of claim **15** wherein each row of the pivot table identifies a custom column, a row of the physical table, and data for the identified custom column of the identified row of the physical data.

17. The computer-readable storage medium of claim **14** wherein the logical table is generated by providing a result set containing physical data retrieved from standard and custom columns of the physical table and the map is used to map the retrieved physical data to the corresponding logical data of the logical table.

18. The computer-readable storage medium of claim **17** wherein the logical table is represented as a dataset object.

19. The computer-readable storage medium of claim **18** including adding tracker tables to the dataset object to log updates to the logical table.

20. The computer-readable storage medium of claim **14** wherein the physical table includes multiple standard tables and the updating includes updating only those standard tables with standard columns corresponding to logical columns that were updated.

21. A computer-readable storage medium containing instructions for controlling a computer system to update data, by a method comprising:

providing a map between physical columns of a physical table and logical columns of a logical table, the physical table being represented a multiple database tables within a database;

providing a result set containing physical data derived from the multiple database tables;

organizing the physical data of the result set into a logical table based on the provided map;

providing an indication of an update to the logical table, the update updating logical columns corresponding to physical columns represented in different database tables;

using the provided map to determine to which columns of which database tables the updated logical columns correspond; and

effecting the update of the determined columns of the database tables.

22. The computer-readable storage medium of claim **21** wherein the database tables correspond to standard tables and a custom table, the custom table being represented by a pivot table.

23. The computer-readable storage medium of claim **22** wherein when a determined column is a custom column of the custom table, updating a row of the pivot table.

24. The computer-readable storage medium of claim **22** wherein each row of the pivot table identifies a custom column, a row of the physical table, and data for the identified custom column of the identified row of the physical data.

25. The computer-readable storage medium of claim **21** wherein the logical table is represented as a dataset object.

26. The computer-readable storage medium of claim **25** including adding tracker tables to the dataset object to log updates to the logical table.

27. The computer-readable storage medium of claim **21** wherein the physical table includes multiple standard tables and the updating includes updating only those standard tables with standard columns corresponding to logical columns that were updated.

* * * * *

Exhibit B

US005742768A

United States Patent [19]

Gennaro et al.

[11] Patent Number: 5,742,768

[45] Date of Patent: Apr. 21, 1998

[54] **SYSTEM AND METHOD FOR PROVIDING AND DISPLAYING A WEB PAGE HAVING AN EMBEDDED MENU**

[75] Inventors: **Giuseppe Gennaro**, Cupertino; **Jake McGowan**, San Jose; **Anne P. Wagner**, Menlo Park; **Kinney Wong**, San Jose; **Benjamin A. Zamora**, Stanford, all of Calif.

[73] Assignee: **Silicon Graphics, Inc.**, Mountain View, Calif.

[21] Appl. No.: **680,836**

[22] Filed: **Jul. 16, 1996**

[51] **Int. Cl.**[6] **G06F 13/00**

[52] **U.S. Cl.** **295/200.33**

[58] **Field of Search** ..... 364/DIG. 1, DIG. 2; 395/761, 762, 326, 352, 353, 354, 355, 356, 357, 358, 200.47, 200.48, 20.33

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,572,643 11/1996 Judson 395/200.48

OTHER PUBLICATIONS

John C. Dhabolt "Re: Help with Menus" http://dejanews.com (Jun. 13, 1996) p. 1.

Primary Examiner—Robert B. Harrell

Attorney, Agent, or Firm—Baker & Botts, L.L.P.

[57] **ABSTRACT**

A method for providing a web page (**26**) having an embedded menu (**46**) to a web browser (**24**) and for displaying the web page (**40**) to a user of the web browser (**24**) are provided. A request for a web page (**20**) is received from a web browser (**24**). In response to the request, a web page (**26**) and an applet (**28**) associated with the web page (**20**) are packaged for transmission to the web browser (**24**). The web page (**26**) and the applet (**28**) are then transmitted to and downloaded by the web browser (**24**). When the web page (**26**) is displayed and the applet (**28**) is executed by the web browser (**24**), the applet (**28**) creates and manages an embedded menu (**46**) in the displayed web page (**40**)under control of the applet (**28**). This embedded menu (**46**) provides a user of the web browser (**24**) with a plurality of links (**48**) through one action in the displayed web page (**40**).

20 Claims, 3 Drawing Sheets

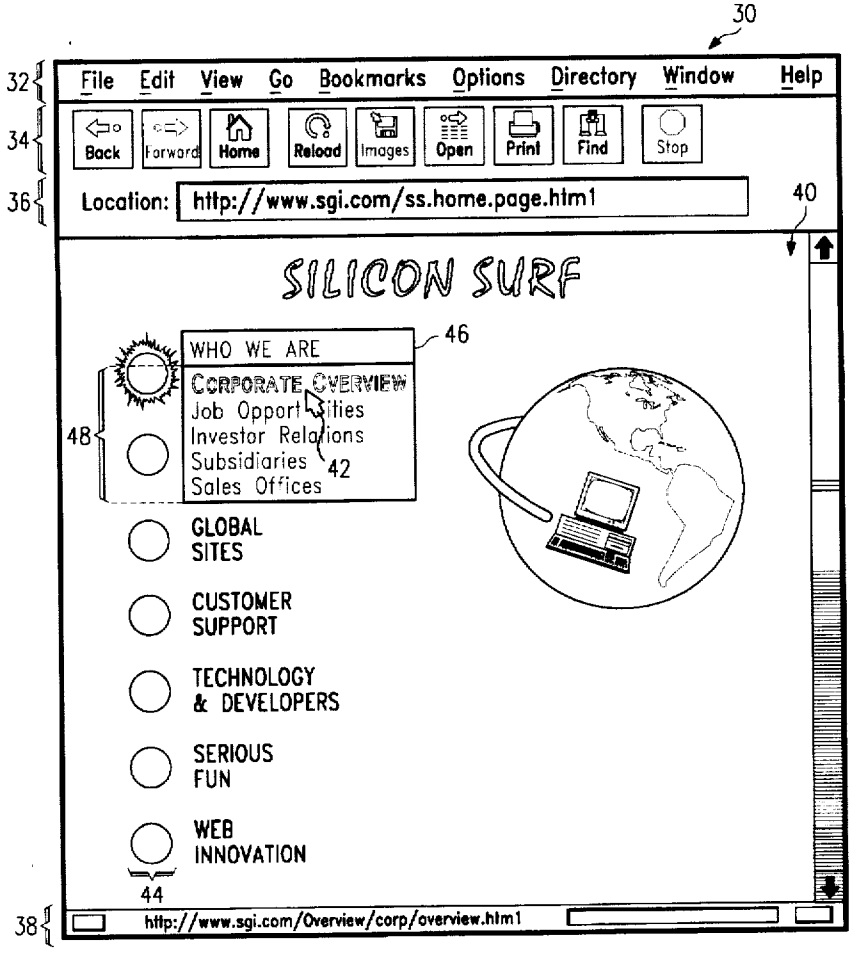


FIG. 1

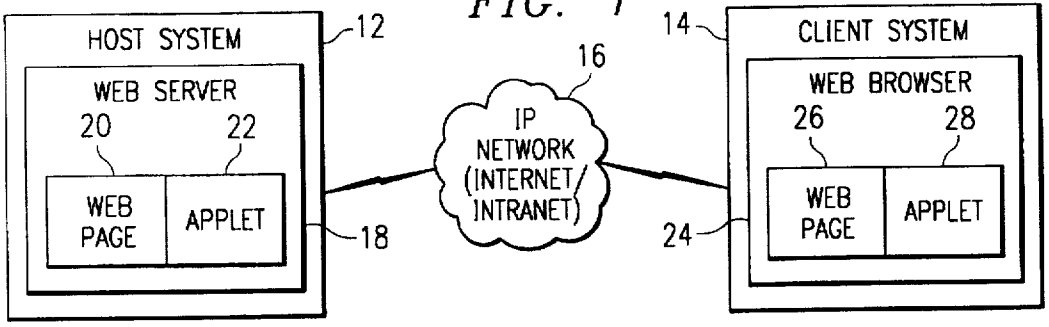

HOST SYSTEM
12
WEB SERVER
20
22
WEB PAGE
APPLET
18
16
IP NETWORK (INTERNET/ INTRANET)
14
CLIENT SYSTEM
WEB BROWSER
26
28
WEB PAGE
APPLET
24

FIG. 2A

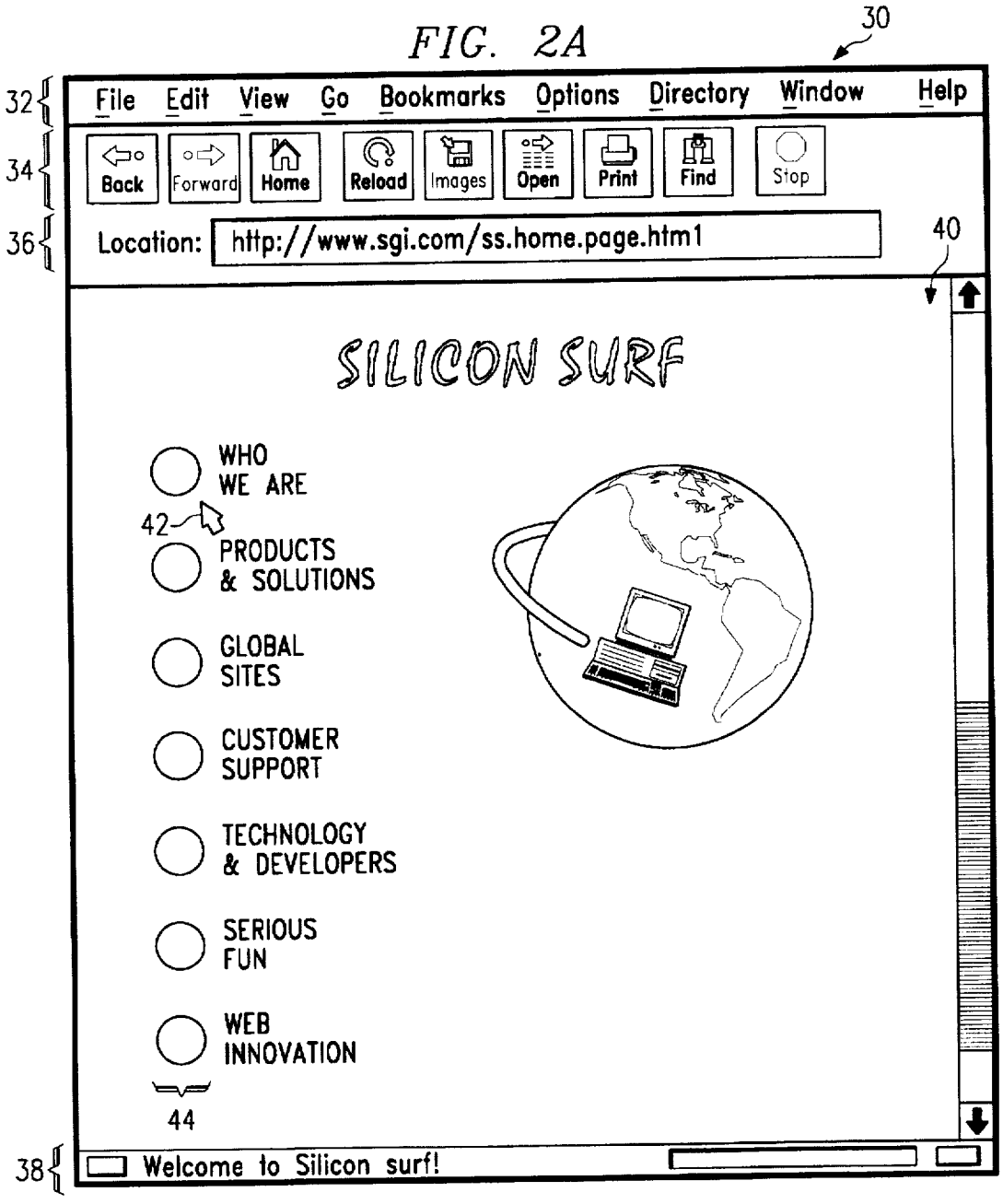
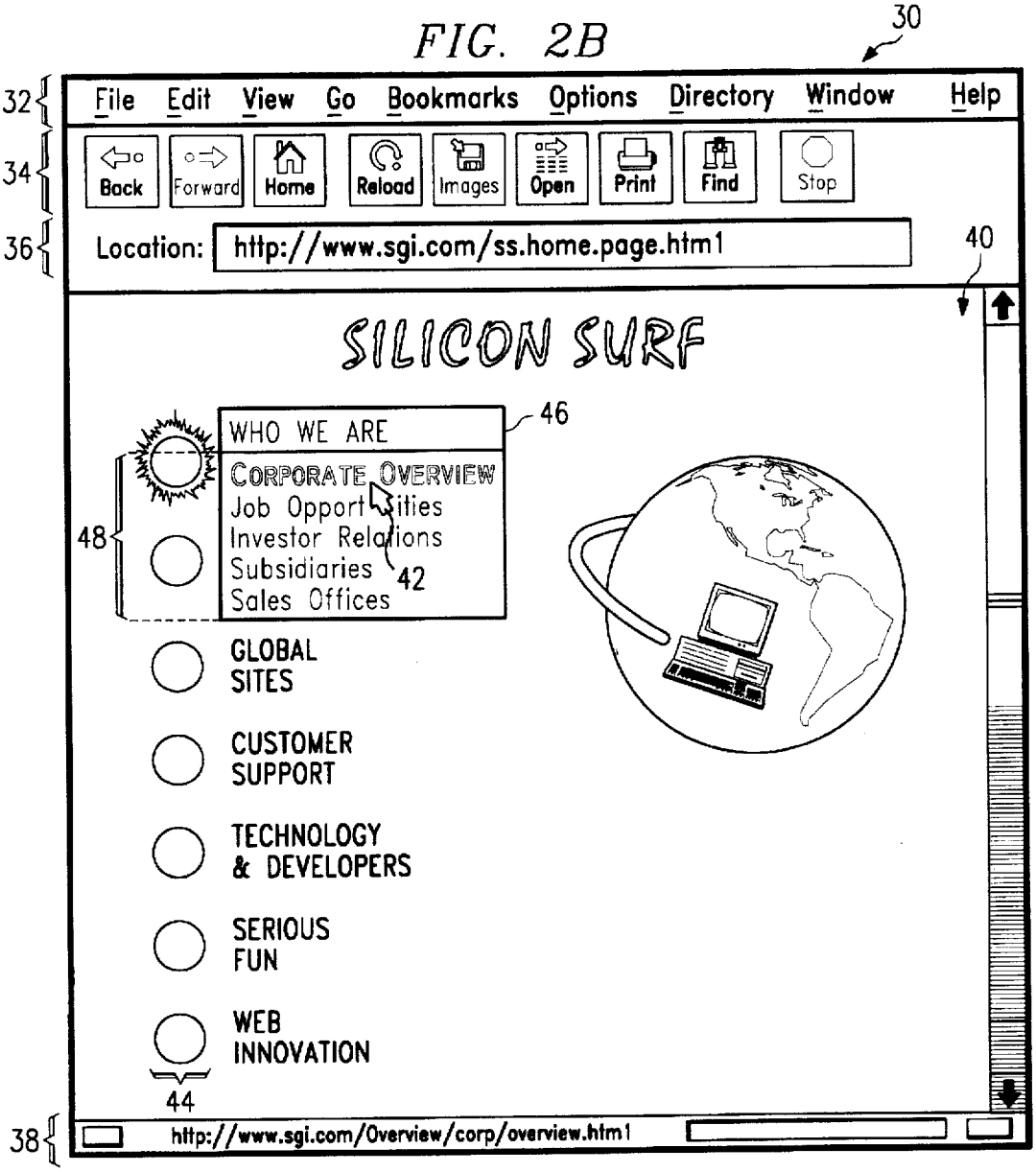

30
32
File Edit View Go Bookmarks Options Directory Window Help
34
Back Forward Home Reload Images Open Print Find Stop
36
Location: http://www.sgi.com/ss.home.page.htm1
40
SILICON SURF
WHO WE ARE
42
PRODUCTS & SOLUTIONS
GLOBAL SITES
CUSTOMER SUPPORT
TECHNOLOGY & DEVELOPERS
SERIOUS FUN
WEB INNOVATION
44
38
Welcome to Silicon surf!

FIG. 2B



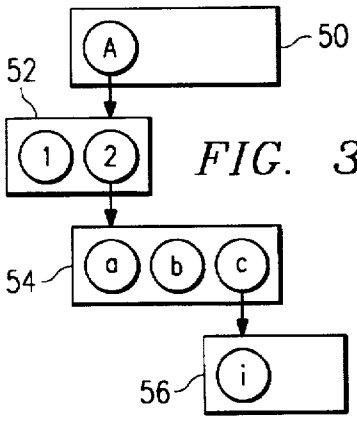


FIG. 3A

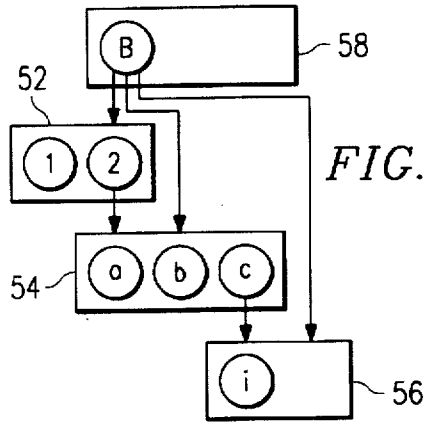
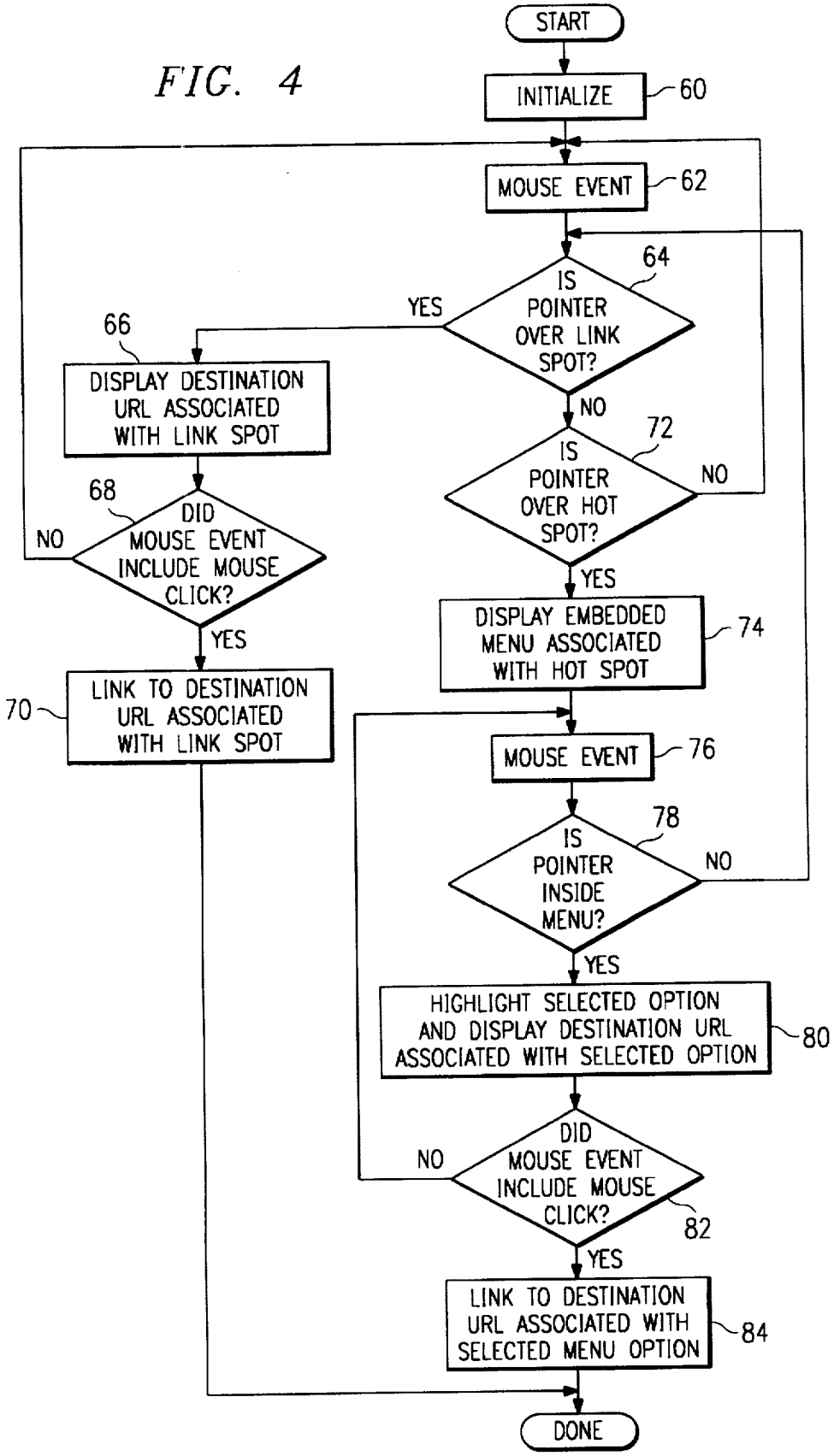


FIG. 3B

FIG. 4


START
INITIALIZE
60
MOUSE EVENT
62
64
IS POINTER OVER LINK SPOT?
YES
66
DISPLAY DESTINATION URL ASSOCIATED WITH LINK SPOT
NO
72
IS POINTER OVER HOT SPOT?
NO
68
DID MOUSE EVENT INCLUDE MOUSE CLICK?
NO
YES
YES
DISPLAY EMBEDDED MENU ASSOCIATED WITH HOT SPOT
74
70
LINK TO DESTINATION URL ASSOCIATED WITH LINK SPOT
MOUSE EVENT
76
78
IS POINTER INSIDE MENU?
NO
YES
HIGHLIGHT SELECTED OPTION AND DISPLAY DESTINATION URL ASSOCIATED WITH SELECTED OPTION
80
DID MOUSE EVENT INCLUDE MOUSE CLICK?
NO
82
YES
LINK TO DESTINATION URL ASSOCIATED WITH SELECTED MENU OPTION
84
DONE

SYSTEM AND METHOD FOR PROVIDING AND DISPLAYING A WEB PAGE HAVING AN EMBEDDED MENU

TECHNICAL FIELD OF THE INVENTION

This invention relates in general to the field of electronic systems, and more particularly to a system and method for providing and displaying a web page having an embedded menu.

BACKGROUND OF THE INVENTION

Web servers and web browsers operating across an IP (internet protocol) network are widely used to provide remote access to information stored on a host system. The public Internet and private intranets are examples of such IP networks and use a communication protocol referred to as the hyper-text transfer protocol (HTTP). The information is commonly packaged as multiple web pages created using a hyper-text markup language (HTML) which can be interpreted by a web browser to generate the display to the user.

In general, URLs (uniform resource locators) are used to identify web pages located on web servers operating on the network. A user of a web browser can request a web page by entering the appropriate URL into the web browser. A request for the selected web page is then transmitted to the web server across the network. The web server receives the request and then packages and transmits the web page back to the web browser for display to the user.

The interaction between web servers and web browsers across the IP network provides a relatively easy and increasingly popular means for accessing remote information. However, the process of navigating through this information in conventional web pages is a linear process. Web pages provide links that correspond one-to-one with other web pages and resources. Thus, one action inside the web page (e.g., a mouse click) can initiate one link to another web page.

One of the means for enhancing a web page is the use of an executable program attached to web page which is downloaded to and executed by the web browser along with the associated web page. These executable programs are commonly referred to as applets and are constructed from a programming language which is executable by the web browser. Once executed by the web browser, the applet provides programmed functionality. For example, the JAVA programming language established by SUN MICROSYSTEMS provides a means for creating JAVA applets which can be attached to web pages to provide enhanced functionality for the displayed web page. One example of a function created by applets is animating an image to produce moving objects on the web page. Applets have also been used to create executable spots in a web page such that graphics on that spot animate when a mouse pointer is moved over the targeted area. An additional function created by applets is to generate and display a separate window on top of a web page in response to a mouse click inside the web page. Such a window can provide a menu bar across the top of the window and provide user options within that window, but it is not within the web page itself. There are, of course, a number of other functions that can be implemented using applets in association with web pages. However, conventional web pages and applets have not altered the linear navigation process.

SUMMARY OF THE INVENTION

In accordance with the present invention, a system and method for providing and displaying a web page having an embedded menu are provided which substantially eliminate or reduce disadvantages and problems associated with previously developed web pages.

According to one aspect of the present invention, a method for providing a web page having an embedded menu to a web browser is provided. This method includes receiving a request for a web page from a web browser. In response to the request, a web page and an applet associated with the web page are packaged for transmission to the web browser. The web page and the applet are then transmitted to the web browser. The applet is operable to create and manage an embedded menu in the displayed web page when the web page is displayed and the applet is executed by the web browser. This embedded menu provides a user of the web browser with a plurality of links through one action in the displayed web page.

According to another aspect of the present invention, a method for displaying a web page having an embedded menu to a user of a web browser is provided. This method includes downloading a web page and an applet from a web server. The web page is then displayed to a user of the web browser, and the applet is executed by the web browser. An embedded menu is created and managed in the displayed web page under control of the applet. This embedded menu provides a user of the web browser with a plurality of links through one action in the displayed web page.

According to a further aspect of the present invention, a host system for providing and a client system for displaying a web page having an embedded menu are provided. The host system includes a data storage device, a memory and a processor and executes a web server for packaging and transmitting the web page and applet. The client system includes a display, a memory and a processor and executes a web browser for downloading the web page and the applet and for displaying the web page and executing the applet. The applets enhance the web page to have an embedded menu that provides a plurality of links through one action in the displayed web page.

Embedding a menu in a web page to allow a user of a web browser to access multiple links through one action in the web page is a technical advantage of the present invention. The web page is enhanced through the use of an applet which creates and manages the embedded menu. A web page having an embedded menu according to the present invention provides an easier and more efficient way to access information from that web page, thus increasing the quality of the web page. In one implementation, the web page has one or more hot spots. When a pointer is positioned over one of these hot spots, a corresponding embedded menu is displayed to provide links to multiple additional web pages. A user can then select a link by positioning the pointer over one of the links and initiating an action such as by clicking a mouse button.

Another technical advantage of the embedded menus of the present invention is the ability to allow a user of a web browser to scan the information content of a web site from an initial displayed web page without linking to new web pages. A user can reposition a pointer over each hot spot to invoke each embedded menu and be provided with multiple links at one or more levels within the web site. For example, an operator of a web server can create a web site for which the initial web page can display, through the use of embedded menus, the overall structure of the web site as well as links to numerous location therein. A user of the web browser can thereby identify and link to desired information more quickly and easily than possible with conventional linear links.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the present invention and advantages thereof may be acquired by referring to the following description taken in conjunction with the accompanying drawings in which like reference numbers indicate like features and wherein:

FIG. **1** is a block diagram of a web server and a web browser in an IP network;

FIGS. **2A** and **2B** illustrate an embedded menu in a web page according to the teachings of the present invention;

FIGS. **3A** and **3B** illustrate a comparison between one-to-one and multiple-to-one correspondence between links and actions in a web page; and

FIG. **4** is a flow chart of a process for managing links and embedded menus in a web page.

DETAILED DESCRIPTION OF THE INVENTION

FIG. **1** is a block diagram of a web server **12** and a web browser **14** in an IP (internet protocol) network **16**. IP network **16** can be, for example, the public Internet or a private intranet, and host system **12** and client system **14** can communicate across IP network **16** using a hyper-text transfer protocol (HTTP).

Host system **12** and client system **14** can be, for example, a personal computer or computer workstation, and generally include a data storage device, a memory device, a processor and a display. The memory device in host system **12** can store code for and the processor can execute a web server **18**. The data storage device in host system **12** can store a web page **20** and an associated applet **22**. Web page **20** can be written in the hyper-text mark-up language (HTML), and applet **22** can be written in an interpretive language such as JAVA. Analogous to host system **12**, the memory device in client system **14** can store code for and the processor can execute a web browser **24**. The memory device in client system **14** can store a downloaded web page **26** and an associated applet **28**. Web browser **24** is an applet-capable web browser and can both display web page **26** and execute applet **28**.

In operation, a user of client system **14** can use web browser **24** in order to transmit a request for web page **20** across IP network **16**. The request can be, for example, a URL (uniform resource locator) for web page **20**. Web server **18** can receive the request from web browser **24** and, in response, can package and transmit web page **26** and applet **28** to web browser **24** across IP network **16**. Web server **18** packages web page **26** and applet **28** based upon web page **20** and applet **22** stored on host system **12**. After downloading web page **26** and applet **28**, web browser **24** can display web page **26** to a user of client system **14** and can execute applet **28**. Applet **28** only needs to be downloaded once and is executed by web browser **24**. When a link is selected in web page **26**, web server **18** is sent a request across IP network **16** and can transmit the selected page.

The execution of applet **28** by web browser **24** can provide enhanced functionality to web page **26**. According to the teachings of the present invention, applet **28** creates and manages one or more embedded menus in the displayed web page **26**. Each embedded menu provides a user of web browser **24** with a plurality of links through one action in the displayed web page **26**.

FIGS. **2A** and **2B** illustrate an embedded menu in a web page according to the teachings of the present invention. As shown in FIG. **2A**, a display window generated by web

browser **24**, indicated generally at **30**, can include a menu bar **32** and a plurality of buttons **34** each providing one of various functions for a user of web browser **24**. Display window **30** also includes a location field **36** which serves a dual function of indicating the URL of the current location and of allowing a user to enter a new destination URL. In the illustrated example, web browser **24** is being used to navigate the public Internet, and the URL shown in location field **36** is the URL of a web page on the world wide web. Display window **30** further includes a status bar **38** that provides information about the operation of web browser **24**. The items in menu bar **32** and buttons **34** and the general layout of display window **30** as shown in FIG. **2A** are common features of the NETSCAPE NAVIGATOR web browser available from NETSCAPE COMMUNICATIONS.

Display window **30** includes a displayed web page, indicated generally at **40**, which is generated by web browser **24** from the downloaded web page **26** and associated applet **28**. Displayed web page **40** provides the user of web browser **24** with the information content accessed from web server **18**. The user generally interacts with display window **30** and displayed web page **40** using a pointer device (e.g., a mouse) which controls the position of a pointer **42** and allows a user to initiate actions (e.g., through a mouse click). According to the teachings of the present invention, displayed web page **40** includes a plurality of hot spots **44** that provide access to embedded menus created and managed by applet **28**. The embedded menus can be accessed by positioning pointer **42** over one of hot spots **44**.

FIG. **2B** shows an embedded menu **46** in displayed web page **40** which has been invoked by positioning of pointer **42** over the upper hot spot **44**. In the illustrated example, selection of the upper hot spot **44** is indicated by highlighting that hot spot **44** with a halo, as shown. Embedded menu **46** includes a banner that matches the text ("WHO WE ARE") that was associated with the selected hot spot **44** in FIG. **2A**. Embedded menu **46** also includes a number of links **48**, each providing a link to another web page or resource. The links **48** provided by embedded menu **46** may or may not be URLs directly accessible without initially passing through the initial displayed web page **40**.

In FIG. **2B**, the "Corporate Overview" link is selected by the positioning of pointer **42** over that portion of embedded menu **46**, and status bar **40** reflects the URL associated with the "Corporate Overview" link. If desired, the user of web browser **24** can link to the "Corporate Overview" information by initiating an action, for example by clicking a mouse button, while pointer **42** is in this position. The user could also move pointer **42** elsewhere in embedded menu **46** to select and initiate one of the other links. In other embodiments, embedded menu **46** can have multiple levels of menus accessible through initial menu options.

When pointer **42** is moved outside of embedded menu **46**, embedded menu **46** will be removed and displayed web page **40** will again look as shown in FIG. **2A**. The user can move pointer **42** over any of hot spots **44** and invoke an associated embedded menu, each of which would provide multiple links to other web pages or resources.

Another technical advantage of the embedded menus of the present invention is the ability to allow a user of web browser **24** to scan the information content that can be linked from an initial displayed web page **40** without linking to new web pages. A user can reposition pointer **42** over each hot spot **44** and be provided with link options within a web site at one or more levels. For example, an operator of web server **18** can create a web site for which the initial web page

can display, through the use of embedded menus, the overall structure of the web site as well as links to a number of locations therein. A user of web browser **24** can thereby identify and link to desired information more quickly and easily than possible with conventional linear links.

FIGS. **3**A and **3**B illustrate a comparison between one-to-one and multiple-to-one correspondence between links and actions in a web page. In FIG. **3**A, web page **50** includes conventional links which correspond on a one-to-one basis with actions in web page **50**. For example, link "A" points to web page **52**, and a user can initiate a link to web page **52** from web page **50** by positioning a mouse pointer on link "A" and clicking the mouse button. Similarly, link "2" in web page **52** points to a web page **54**, and link "c" in web page **54** points to a web page **56**. Link "1" in web page **52**, links "a" and "b" in web page **54** and link "i" in web page **56** point to other web pages, which are not shown. In order to navigate web pages **52**, **54** and **56** from a starting point of web page **50**, a user is required to travel linearly through links "B", "c" and "i" in web pages **50**, **52** and **54**. This linear navigation requires the user to wait as each link is processed. If using conventional links, an operator of web page **50** would need to add two links to web page **50** on order to allow a user to initiate a direct link to all three web pages **52**, **54** and **56**. The user is still only able to access one link for each action in web page **50**.

Web page **58** of FIG. **3**B has an embedded menu according to the teachings of the present invention that provides multiple-to-one correspondence between links and actions in web page **58**. Web page **58** provides a user with a much easier and more efficient access to web pages **52**, **54** and **56**. Web page **52** has an associated applet that creates and manages an embedded menu accessible through hot spot "B". When the embedded menu is invoked, the embedded menu provides the user with links to web pages **52**, **54**, and **56**. Through one action in web page **58**, the user can access all three links. This allows the user to more quickly and easily navigate web pages **58**, **52**, **54** and **56**. For example, the user does not have to wait for multiple links to be processed in order to reach web page **56** as is required for the linear links described above.

FIG. **4** is a flow chart of a process for managing link spots and hot spots in a web page according to the teachings of the present invention. With respect to this process, the term "link spots" is used to refer to conventional one-to-one links within a web page. In general, this process is performed by an applet executed by a web browser in along with the display of an associated web page.

In step **60**, the web page and applet are initialized such that they are displayed and executed by the web browser. In step **62**, the web browser recognizes a mouse event initiated by a user of the web browser. In step **64**, the applet checks whether the mouse pointer is over a link spot in the web page. If so, in step **66**, the applet displays the destination URL associated with that link spot. In step **68**, the applet checks whether the mouse event included a mouse click. If so, in step **70**, the applet initiates a link through the web browser to the destination URL associated with the link spot. At this point, a new web page associated with the destination URL will be loaded which may or may not have an associated applet. If no mouse click occurred, the applet follows step **68** by returning to step **62** and obtaining information about the next mouse event.

If, in step **64**, the mouse pointer is not over a link spot, then, in step **72**, the applet determines whether the mouse pointer is over a hot spot. If not, the applet returns to step **62** and waits for information about the next mouse event. If the mouse pointer is over a hot spot, then, in step **74**, the applet displays the embedded menu associated with that hot spot.

After displaying the embedded menu, the applet waits, in step **78**, for information about the next mouse event. If the mouse pointer moves outside of the embedded menu, the applet returns to step **64**. If the mouse pointer remains inside the embedded menu, then, in step **80**, the applet highlights the selected link within the embedded menu and displays the destination URL associated with the selected link. In step **82**, the applet determines whether the mouse event included a mouse click. If not, the applet returns to step **76** to obtain information about the next mouse event. If the mouse event did include a mouse click, then, in step **84**, the applet initiates a link to the destination URL associated with the selected menu link. At this point, a new web page associated with the destination URL will be loaded by the web browser which may or may not have an associated applet.

One implementation of an applet according to the teachings of the present invention uses the JAVA programming language established by SUN MICROSYSTEMS. The JAVA programming language provides features to allow frame and window classes to be defined where the frame class provides a separate window created and displayed on top of the web page, and the window class provides an ability to have a menu bar across the top of that separate window. However, the JAVA language does not contemplate embedding a menu in a web page to provide multiple links from one action in the web page. In order to create such an embedded menu using a JAVA applet, the present invention defines a new JAVA class which implements the embedded menu.

The following TABLE provides an outline of the JAVA applet and new class of this implementation of the present invention.

TABLE

Applet and Embedded Menu Class
APPLET
Import Java elements
Import Embedded Menu Class
Define variables
Define defaults for embedded menus
(Main loop)
Obtain hot spots and images, overwriting defaults of they exist
Load default menu parameters
Obtain embedded menu for each hot spot
Define default links
Obtain additional links
Draw and cache images
Reset flag for mouse click
Check whether mouse click inside embedded menu or on link spot
EMBEDDED MENU CLASS
Import Java elements
Define variables
Construct menus
Set menu colors
Set menu title
Highlight appropriate menu item
Draw embedded menus, items and border

Although the present invention has been described in detail, it should be understood that various changes, substitutions and alterations can be made hereto without departing from the spirit and scope of the invention as defined by the appended claims.

What is claimed is:

1. A method for providing a web page having an embedded menu to a web browser, the method comprising:

receiving a request for a web page from a web browser;

packaging the web page and an applet associated with the web page for transmission to the web browser; and

transmitting the web page and the applet to the web browser;

wherein the applet is operable to create and manage an embedded menu in a displayed web page when the web page is displayed and the applet is executed by the web browser, the embedded menu providing a user of the web browser with a plurality of links through one action in the displayed web page.

2. The method of claim **1**, wherein packaging the applet comprises packaging an applet that creates and manages a pop-up menu.

3. The method of claim **2**, wherein the applet creates and manages a pop-up menu that is invoked by positioning a pointer over a hot spot in the displayed web page.

4. The method of claim **2**, wherein the applet creates and manages a pop-up menu which has a plurality of menu levels.

5. The method of claim **1**, wherein receiving a request comprises receiving a request across the public Internet.

6. The method of claim **1**, wherein receiving a request comprises receiving a request across a private intranet.

7. The method of claim **1**, wherein packaging the applet comprises a JAVA applet having a definition for an embedded menu class.

8. The method of claim **1**, wherein the request for the web page is received across a private intranet.

9. A method for displaying a web page having an embedded menu to a user of a web browser, the method comprising:

downloading a web page and an applet transmitted by a web server;

displaying the web page to a user of the web browser; and

executing the applet, the applet creating and managing an embedded menu in the displayed web page under control of the applet, the embedded menu providing a user of the web browser with a plurality of links through one action in the displayed web page.

10. The method of claim **9**, wherein creating and managing the embedded menu comprises creating and managing a pop-up menu.

11. The method of claim **10**, wherein creating and managing the embedded menu comprises creating and managing a pop-up menu which is invoked by positioning a pointer over a hot spot in the displayed web page.

12. The method of claim **10**, wherein creating and managing the embedded menu comprises creating and managing a pop-up menu which has a plurality of menu levels.

13. The method of claim **9**, wherein downloading a web page and an applet comprises downloading the web page and the applet across the public Internet.

14. The method of claim **9**, wherein downloading a web page and an applet comprises downloading the web page and the applet across a private intranet.

15. The method of claim **9**, wherein downloading the applet comprises downloading a JAVA applet having a definition for an embedded menu class.

16. A host system executing a web server to provide a web page having an embedded menu to a web browser, the host system comprising:

a data storage device storing a web page and an associated applet;

wherein the associated applet, when executed, can create and manage an embedded menu in a displayed web page;

a memory device storing code for the web server; and

a processor coupled to the data storage device and to the memory device, the processor executing code for the web server such that the web server is operable to:

receive a request for the web page from a web browser;

package the web page and the applet for transmission to the web server; and

transmit the web page and the applet to the web browser;

such that the applet creates and manages an embedded menu in the displayed web page when the web page is displayed and the applet is executed by the web browser, the embedded menu providing a user of the web browser with a plurality of links through one action in the displayed web page.

17. The host system of claim **16**, wherein the embedded menu is a pop-up menu.

18. The host system of claim **17**, wherein the pop-up menu is invoked by positioning a pointer over a hot spot in the displayed web page.

19. The host system of claim **16**, wherein the pop-up menu has a plurality of menu levels.

20. The host system of claim **16**, wherein the request for the web page is received across the public Internet.

* * * * *

Exhibit C

US005644737A

United States Patent [19]

Tuniman et al.

[11] Patent Number: 5,644,737

[45] Date of Patent: Jul. 1, 1997

[54] **METHOD AND SYSTEM FOR STACKING TOOLBARS IN A COMPUTER DISPLAY**

[75] Inventors: **David Charles Tuniman**; **Vinod Anantharaman**, both of Redmond; **Michael Halvar Jansson**, Bellevue, all of Wash.

[73] Assignee: **Microsoft Corporation**, Redmond, Wash.

[21] Appl. No.: **466,611**

[22] Filed: **Jun. 6, 1995**

[51] **Int. Cl.**[6] **G06F 5/01**; G06F 1/00

[52] **U.S. Cl.** **395/352**; 395/348

[58] **Field of Search** 395/155, 156, 395/157, 159, 161, 351, 348, 352, 354

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,500,936	3/1996	Allen et al.	395/156
5,506,952	4/1996	Choy et al.	395/156

OTHER PUBLICATIONS

Perfectoffice® Desktop Application Director (Trademark of Novell, Inc.), 1994, pp. 1–15.

Microsoft Word Ver. 6.0c (Trademark of Microsoft Corporation), 1994, pp. 1–2.

Primary Examiner—Phu K. Nguyen

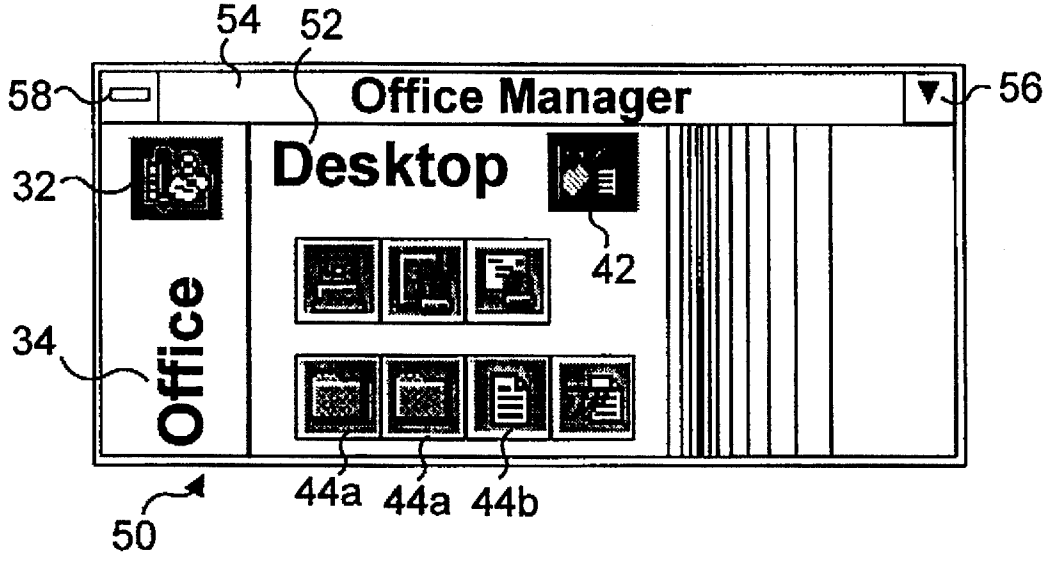
Assistant Examiner—Cliff N. Vo

Attorney, Agent, or Firm—Ronald M. Anderson

[57] **ABSTRACT**

A plurality of toolbars that include graphic objects, which can be selected by the user, are arranged in a stack. Only the graphic objects on one or more selected toolbars are displayed. The user can selectively choose a toolbar that has graphic objects currently hidden by the selected toolbar(s), causing one or more of the toolbars to move aside, exposing the group of graphic objects associated with the toolbar newly selected by the user. Movement of the toolbar(s) to disclose the graphic objects on the newly selected toolbar is preferably accomplished by animating the toolbar(s) to slide to different positions, so that the graphic objects or buttons on the newly selected toolbar are displayed. For added realism, the animation sequence used to disclose graphic objects on a selected toolbar includes an audible sound and a "bump" as the toolbar(s) reach a rest position.

33 Claims, 9 Drawing Sheets



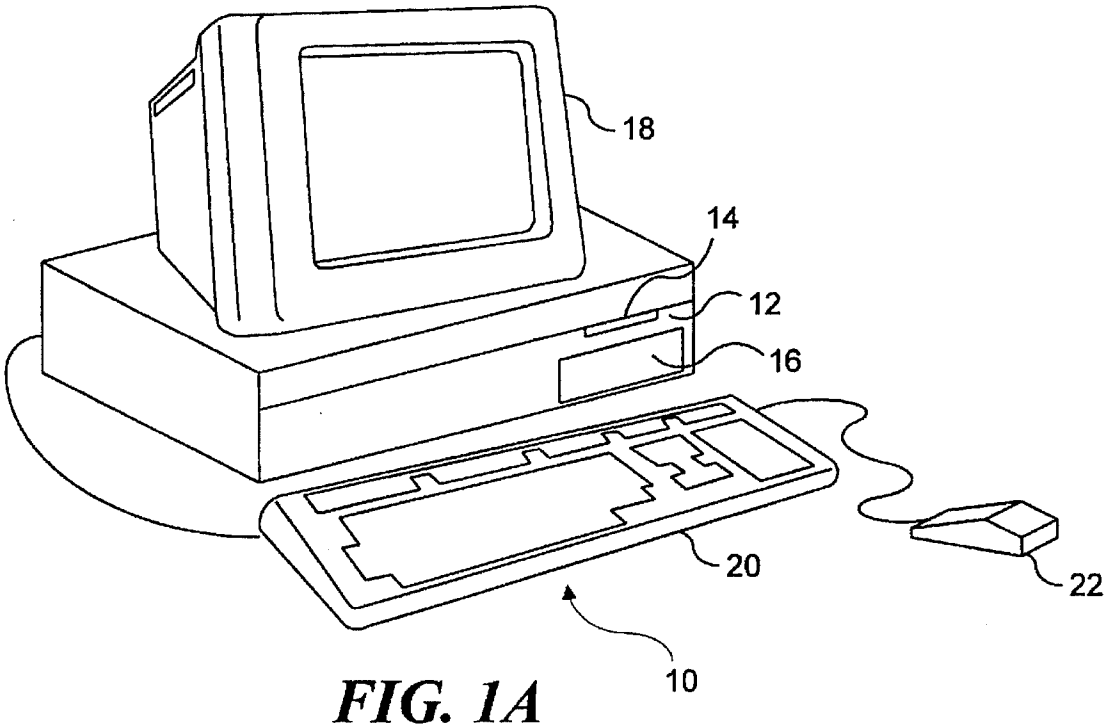

18
14
12
16
20
22
10

FIG. 1A

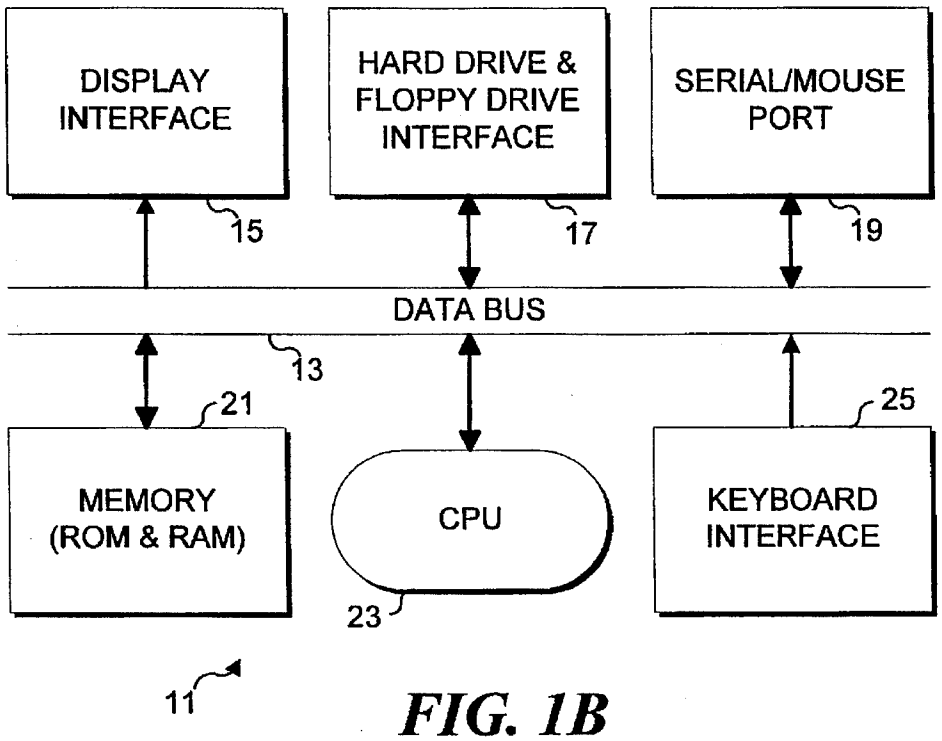

DISPLAY INTERFACE
HARD DRIVE & FLOPPY DRIVE INTERFACE
SERIAL/MOUSE PORT
15
17
19
DATA BUS
13
21
25
MEMORY (ROM & RAM)
CPU
KEYBOARD INTERFACE
23
11

FIG. 1B

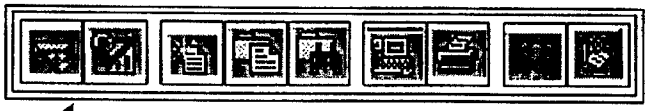


FIG. 2 (PRIOR ART)

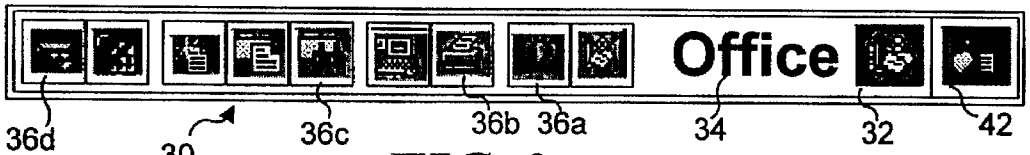


FIG. 3

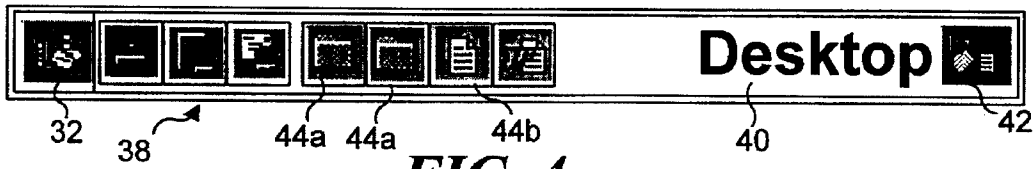


FIG. 4

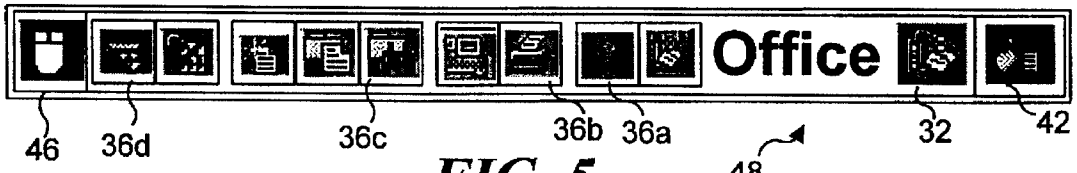


FIG. 5

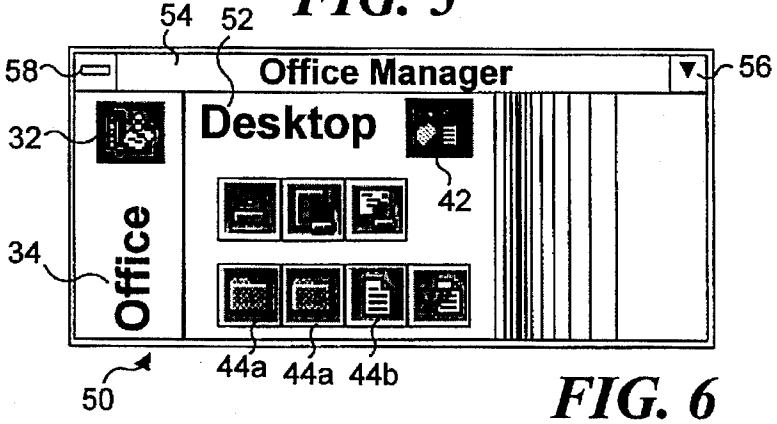


FIG. 6

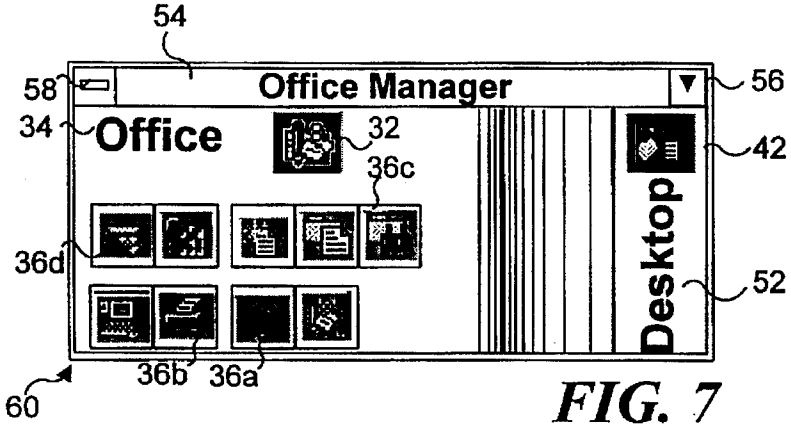


FIG. 7

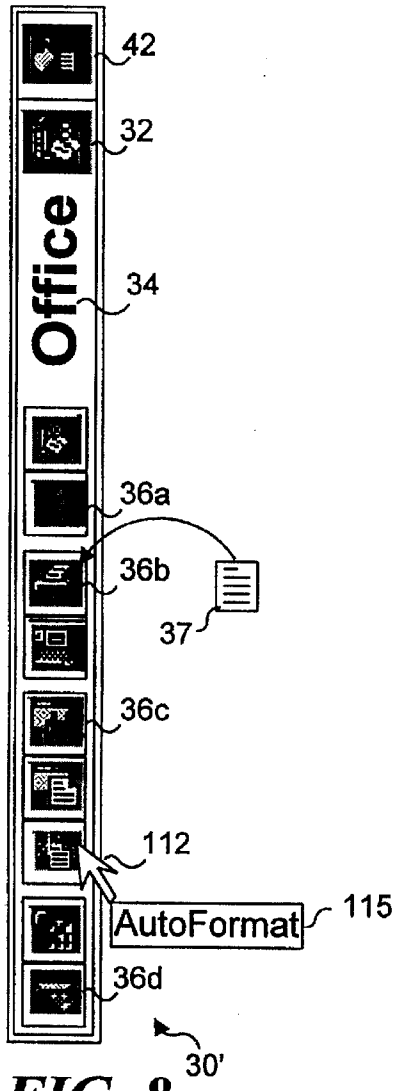


FIG. 8

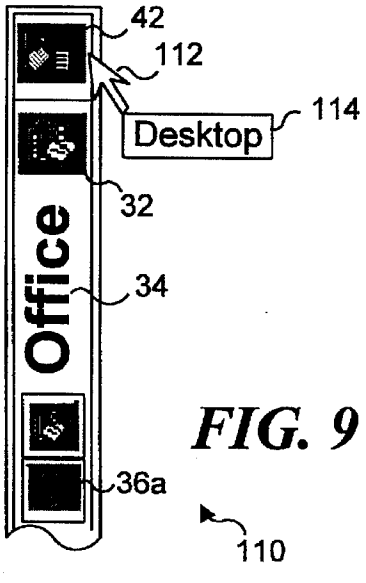


FIG. 9

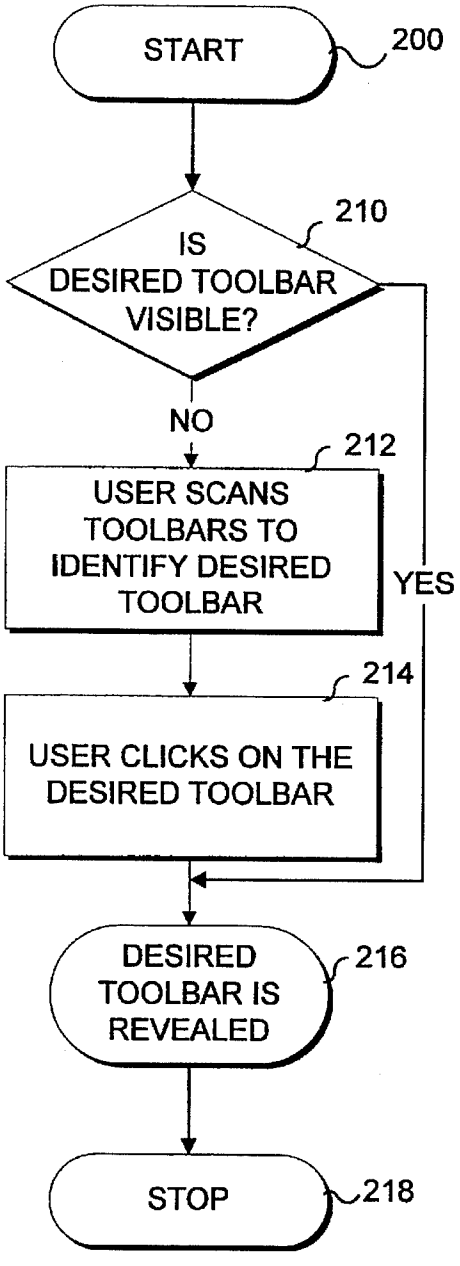


FIG. 14

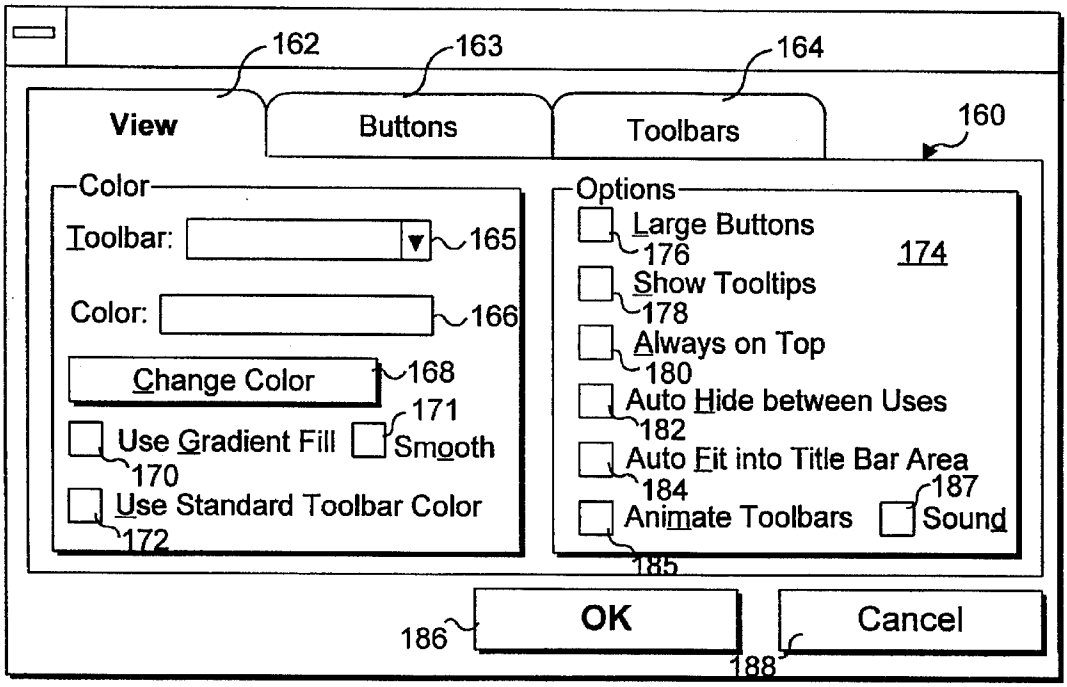

162
163
164
View
Buttons
Toolbars
160
Color
Toolbar:
165
Color:
166
Change Color
168
171
Use Gradient Fill
Smooth
170
Use Standard Toolbar Color
172
Options
Large Buttons
176
174
Show Tooltips
178
Always on Top
180
Auto Hide between Uses
182
Auto Fit into Title Bar Area
184
187
Animate Toolbars
Sound
185
186
OK
Cancel
188

FIG. 10

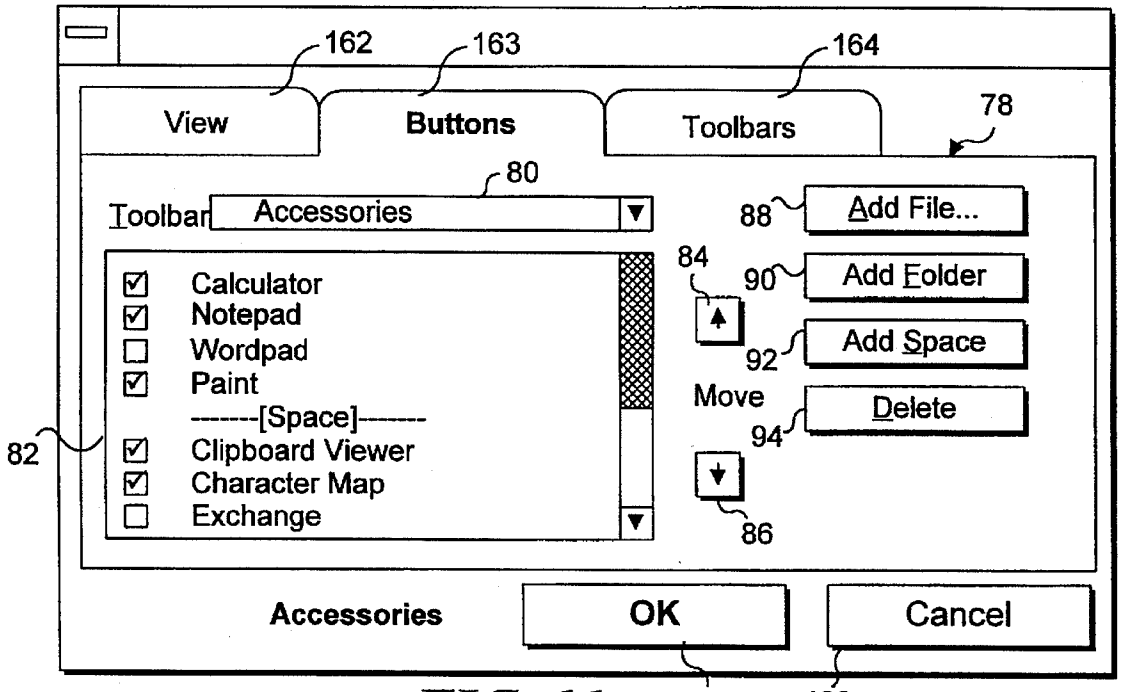

162
163
164
View
Buttons
Toolbars
78
80
Toolbar
Accessories
88
Add File...
84
90
Add Folder
92
Add Space
Move
Delete
94
86
Calculator
Notepad
Wordpad
Paint
-------[Space]-------
Clipboard Viewer
Character Map
Exchange
82
Accessories
OK
Cancel
186
188

FIG. 11

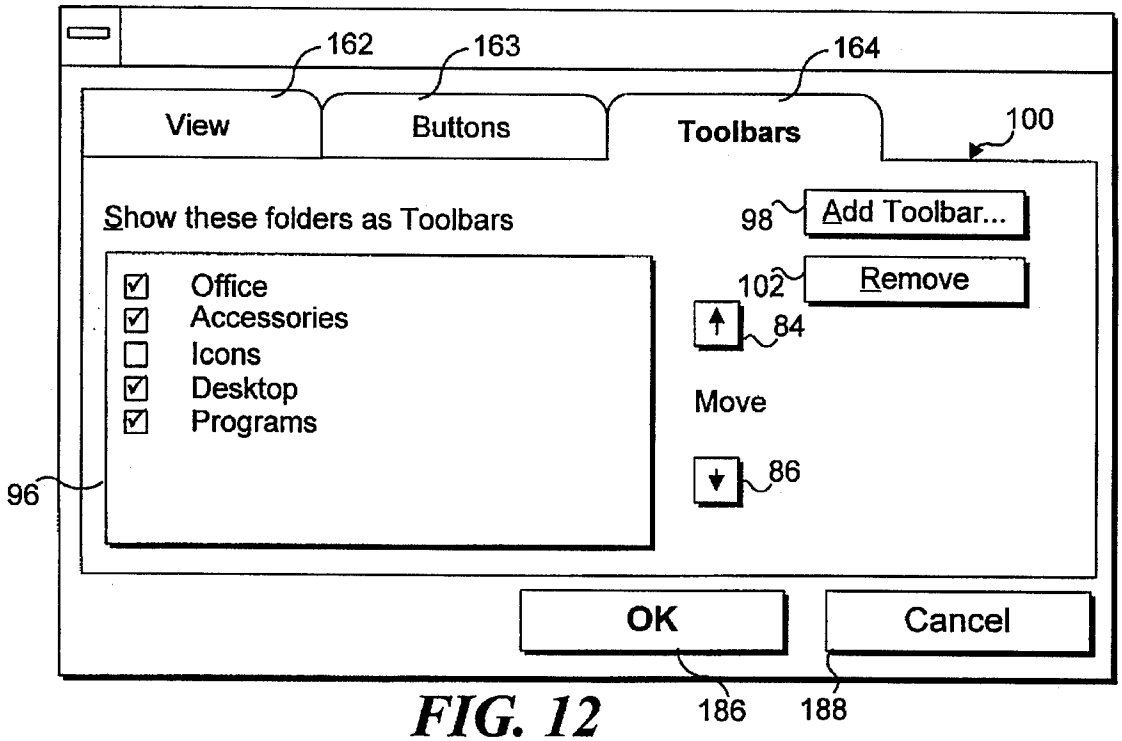

162
163
164
View
Buttons
Toolbars
100
Show these folders as Toolbars
98
Add Toolbar...
102
Remove
84
Office
Accessories
Icons
Desktop
Programs
Move
86
96
OK
Cancel
186
188

FIG. 12

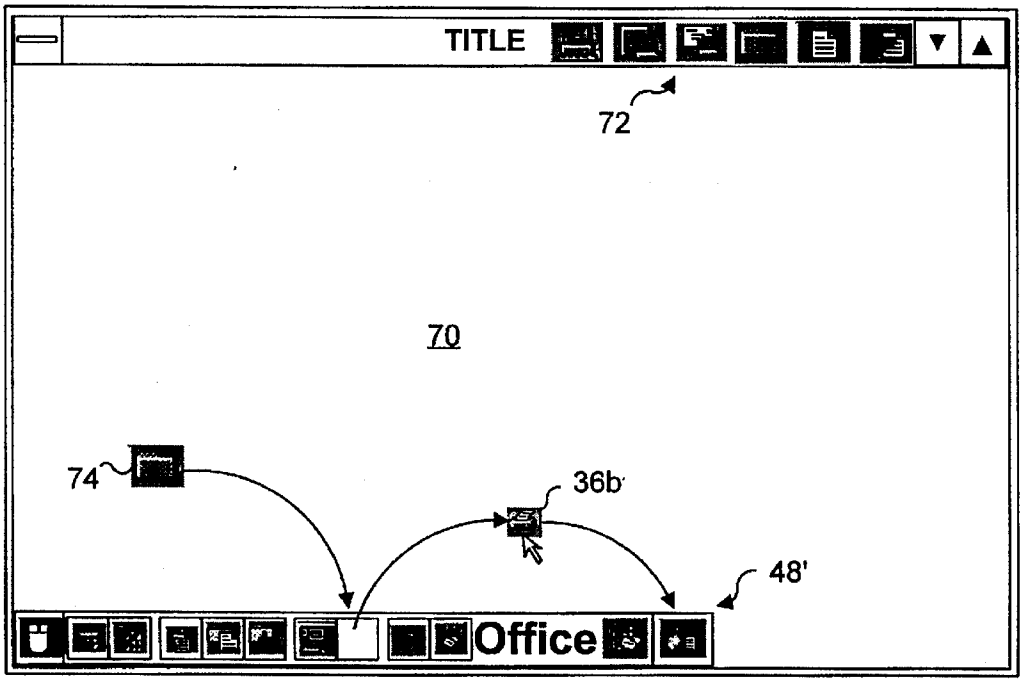

TITLE
72
70
74
36b
48'
Office

FIG. 13

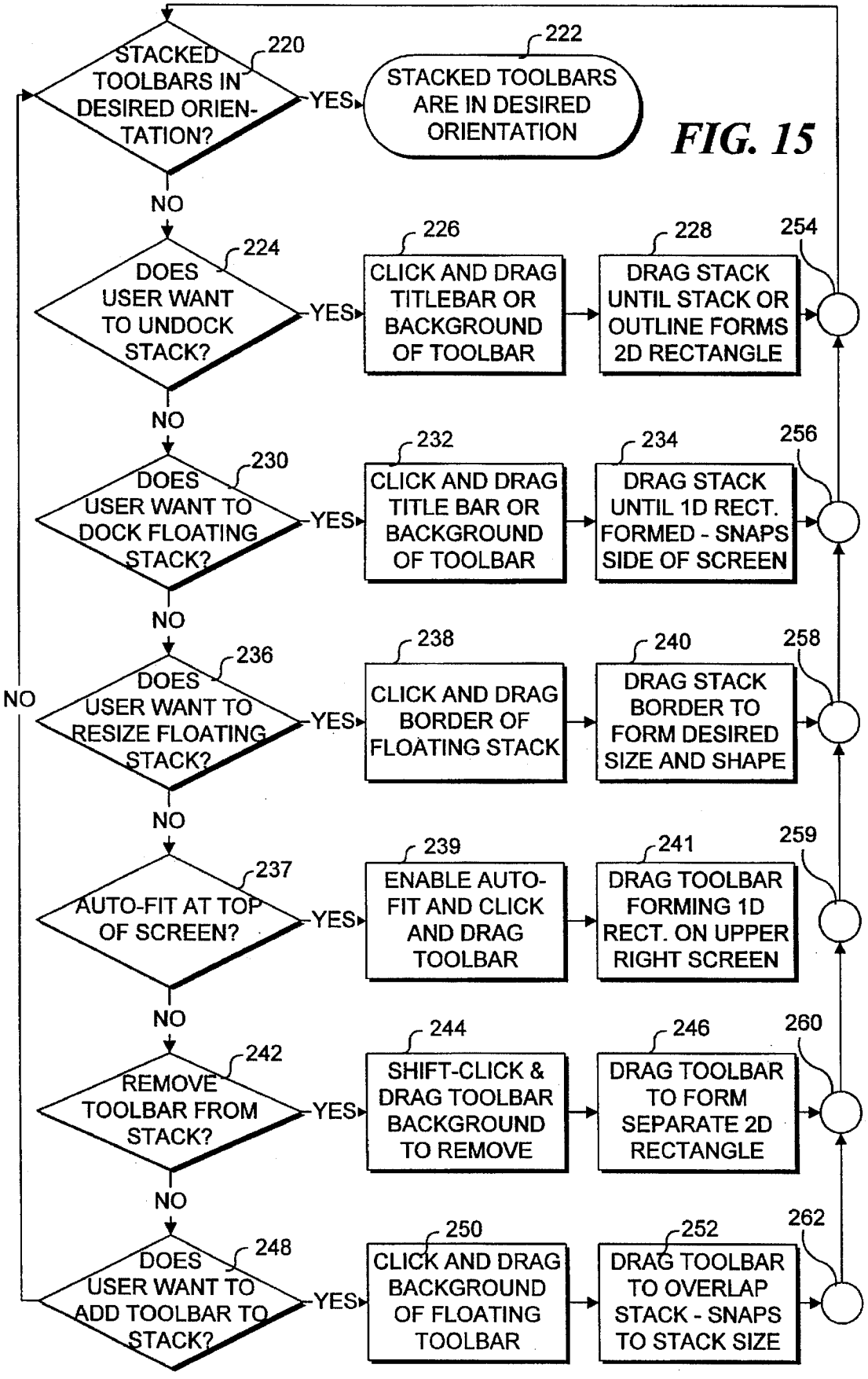

STACKED TOOLBARS IN DESIRED ORIEN-TATION?
220
YES
222
STACKED TOOLBARS ARE IN DESIRED ORIENTATION
NO
DOES USER WANT TO UNDOCK STACK?
224
YES
226
CLICK AND DRAG TITLEBAR OR BACKGROUND OF TOOLBAR
228
DRAG STACK UNTIL STACK OR OUTLINE FORMS 2D RECTANGLE
254
NO
DOES USER WANT TO DOCK FLOATING STACK?
230
YES
232
CLICK AND DRAG TITLE BAR OR BACKGROUND OF TOOLBAR
234
DRAG STACK UNTIL 1D RECT. FORMED - SNAPS SIDE OF SCREEN
256
NO
DOES USER WANT TO RESIZE FLOATING STACK?
236
YES
238
CLICK AND DRAG BORDER OF FLOATING STACK
240
DRAG STACK BORDER TO FORM DESIRED SIZE AND SHAPE
258
NO
NO
AUTO-FIT AT TOP OF SCREEN?
237
YES
239
ENABLE AUTO-FIT AND CLICK AND DRAG TOOLBAR
241
DRAG TOOLBAR FORMING 1D RECT. ON UPPER RIGHT SCREEN
259
NO
REMOVE TOOLBAR FROM STACK?
242
YES
244
SHIFT-CLICK & DRAG TOOLBAR BACKGROUND TO REMOVE
246
DRAG TOOLBAR TO FORM SEPARATE 2D RECTANGLE
260
NO
DOES USER WANT TO ADD TOOLBAR TO STACK?
248
YES
250
CLICK AND DRAG BACKGROUND OF FLOATING TOOLBAR
252
DRAG TOOLBAR TO OVERLAP STACK - SNAPS TO STACK SIZE
262

FIG. 15

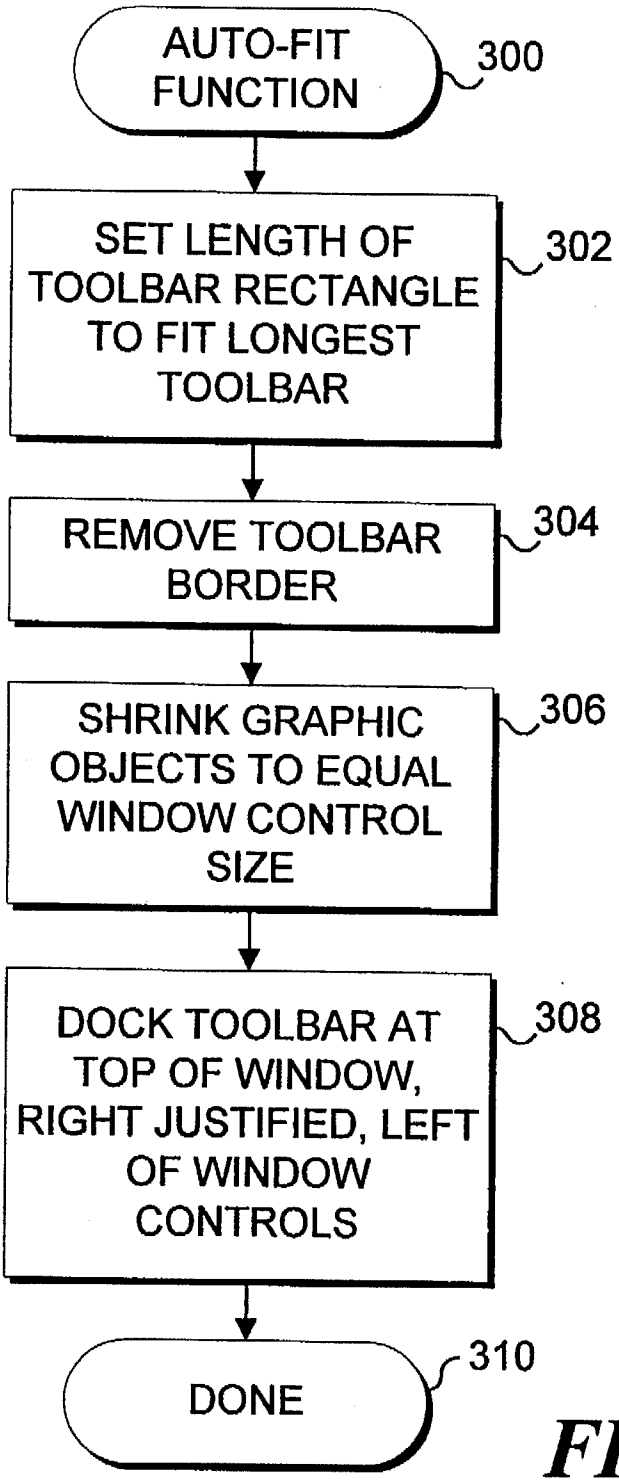

AUTO-FIT FUNCTION
300
SET LENGTH OF TOOLBAR RECTANGLE TO FIT LONGEST TOOLBAR
302
REMOVE TOOLBAR BORDER
304
SHRINK GRAPHIC OBJECTS TO EQUAL WINDOW CONTROL SIZE
306
DOCK TOOLBAR AT TOP OF WINDOW, RIGHT JUSTIFIED, LEFT OF WINDOW CONTROLS
308
DONE
310

FIG. 16

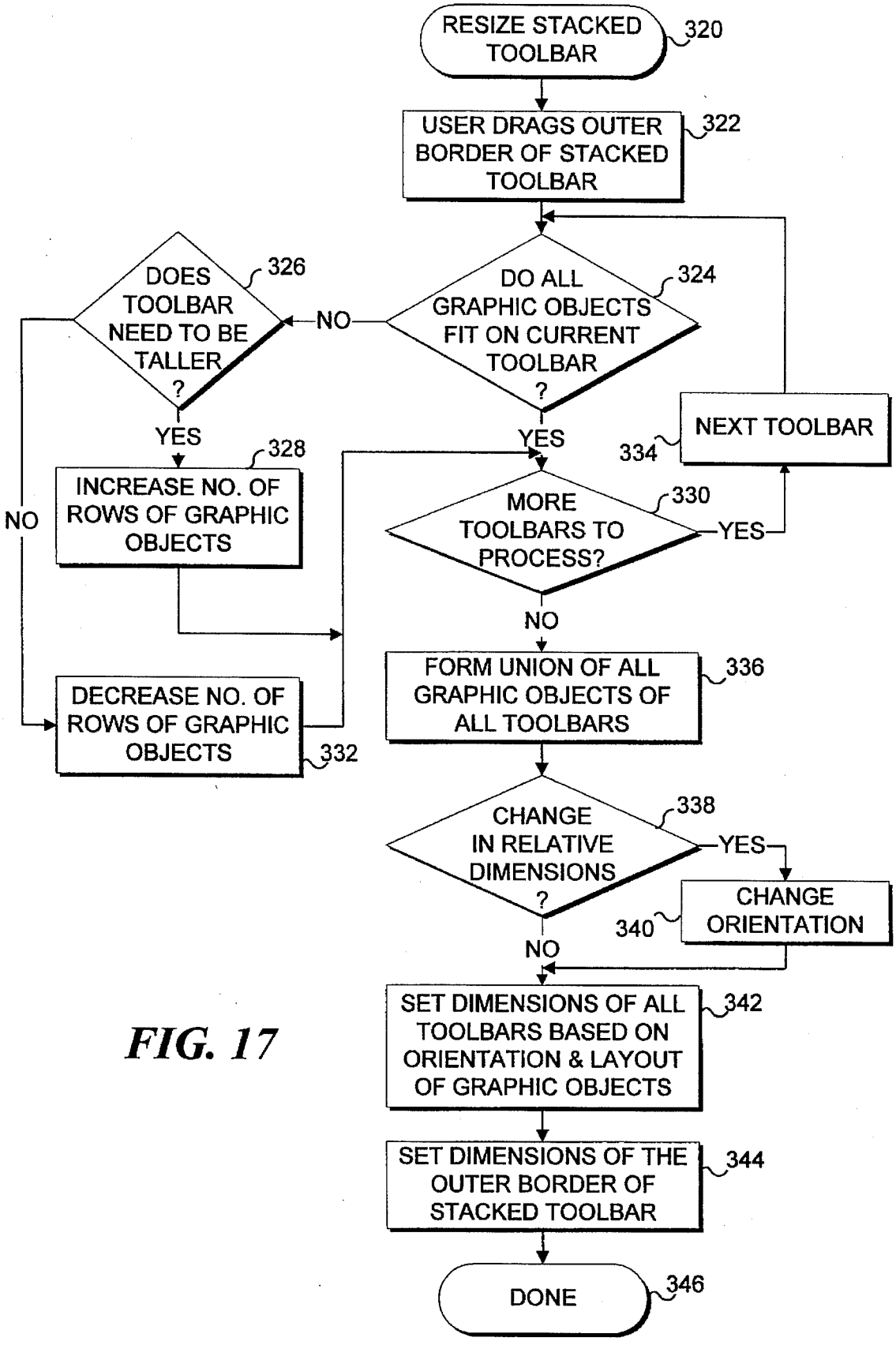

RESIZE STACKED TOOLBAR
320
USER DRAGS OUTER BORDER OF STACKED TOOLBAR
322
DO ALL GRAPHIC OBJECTS FIT ON CURRENT TOOLBAR ?
324
NO
DOES TOOLBAR NEED TO BE TALLER ?
326
YES
INCREASE NO. OF ROWS OF GRAPHIC OBJECTS
328
NO
DECREASE NO. OF ROWS OF GRAPHIC OBJECTS
332
YES
NEXT TOOLBAR
334
MORE TOOLBARS TO PROCESS?
330
YES
NO
FORM UNION OF ALL GRAPHIC OBJECTS OF ALL TOOLBARS
336
CHANGE IN RELATIVE DIMENSIONS ?
338
YES
CHANGE ORIENTATION
340
NO
SET DIMENSIONS OF ALL TOOLBARS BASED ON ORIENTATION & LAYOUT OF GRAPHIC OBJECTS
342
SET DIMENSIONS OF THE OUTER BORDER OF STACKED TOOLBAR
344
DONE
346

FIG. 17

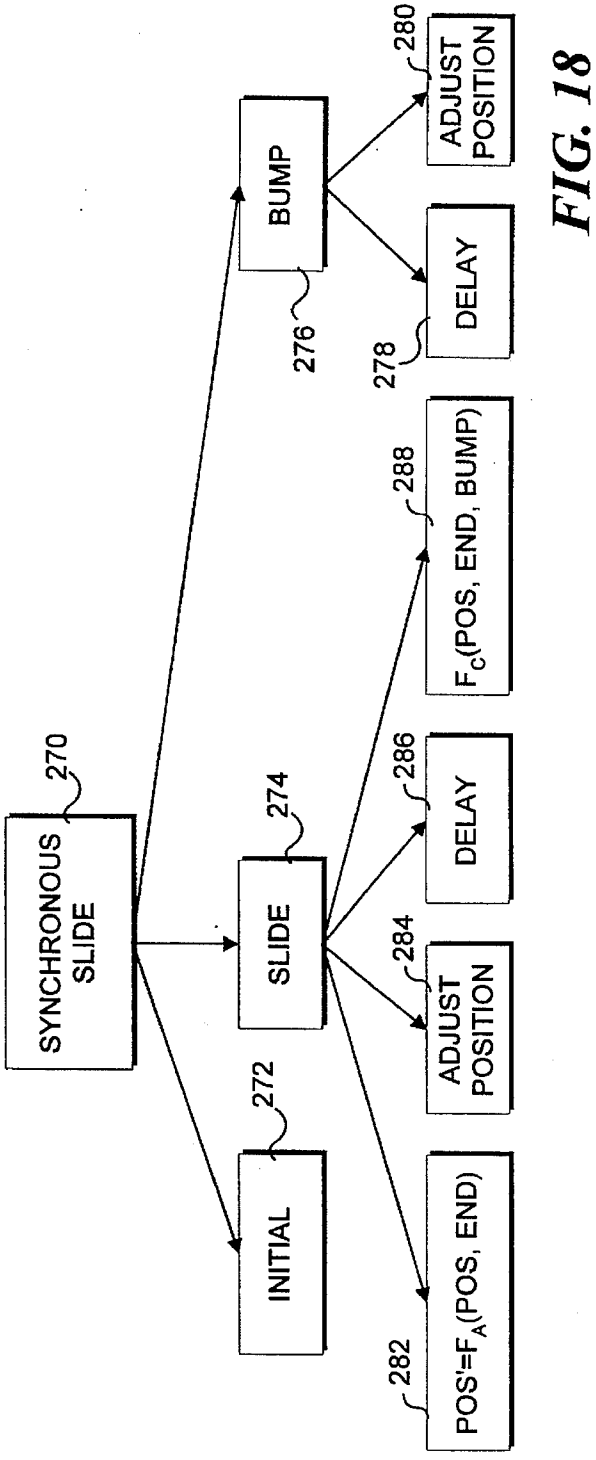

SYNCHRONOUS SLIDE
270
INITIAL
272
SLIDE
274
BUMP
276
DELAY
278
ADJUST POSITION
280
POS'=F_A(POS, END)
282
ADJUST POSITION
284
DELAY
286
F_C(POS, END, BUMP)
288

FIG. 18

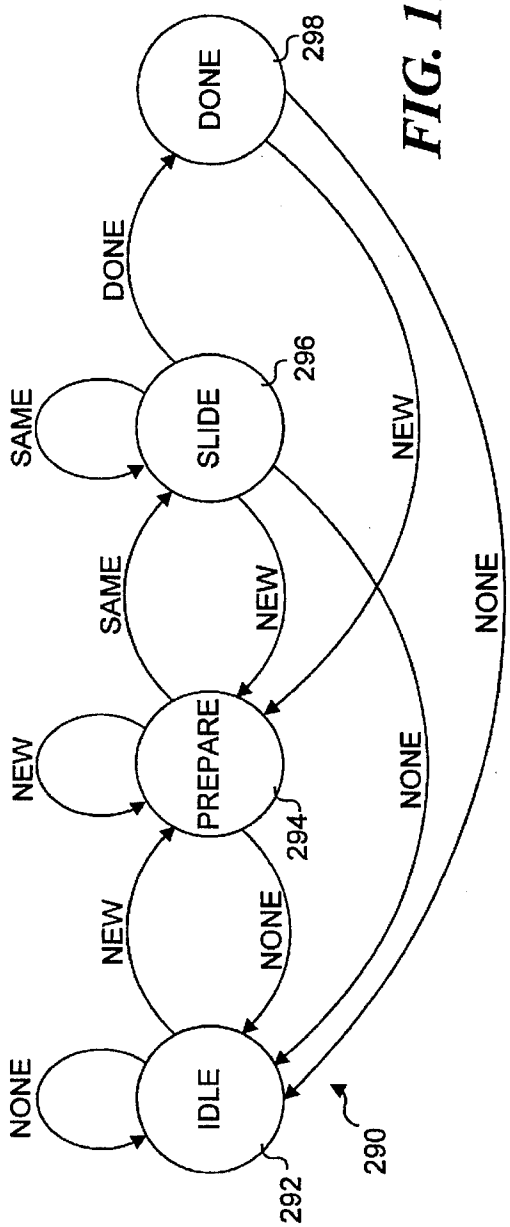

IDLE
PREPARE
SLIDE
DONE
294
296
298
290
292
NONE
NEW
SAME
DONE

FIG. 19

METHOD AND SYSTEM FOR STACKING TOOLBARS IN A COMPUTER DISPLAY

FIELD OF THE INVENTION

This invention generally pertains to a method and system for displaying graphic controls on a computer screen, and more specifically, to the configuration and properties of toolbars comprising a plurality of graphic icons that are displayed for interactive control by a user.

BACKGROUND OF THE INVENTION

Graphics user interface operating systems, such as Microsoft Corporation's WINDOWS™ and related products, have greatly improved the ease with which tasks can be accomplished on the computer. Instead of requiring the user to remember the file name and path of an application program in order to execute it, as has conventionally been the case in text-based operating systems, in a graphic operating system, the user need only activate an icon that represents the application. The graphic icon is activated by positioning a cursor over it and then "double-clicking" on the icon with a pointing device that is used to control the cursor. Since the properties assigned to the graphic icon link it to the executable file and specify its complete directory path, there is no need for the user to recall that information after the icon properties are initially setup to run the program.

Typically, a user will arrange a number of frequently used applications in different groups based on subject matter or common task relationship. Within a group, each application is represented by a graphic icon. In certain desktop shells, the graphic icons representing different applications are arranged in panels of buttons that can be configured in different rectangular shapes by dragging on a side of the panel with the cursor using the pointing device.

Graphic icons are also commonly used within applications to represent different tools, controls, commands, macros, or procedures. For example, Microsoft Corporation's Word for Windows includes several toolbars, each comprising a plurality of different graphic icon buttons that can be selected to carry out various tasks in the word processing system. The user can customize a toolbar or build a new one by adding or deleting graphic icon buttons. The appearance of any graphic icon button can be altered by editing the bitmap that appears on it in an icon editing utility. Toolbars are thus well known in the art and frequently used because they enable a user to immediately access tools within applications, just as the panels of graphic icon buttons that represent different applications or programs enable those applications to be immediately selected on the desktop of the graphic user interface.

However, there is a limitation on the number of graphic icons that can be available to the user. Each time that a toolbar is selected by the user to be left open on the computer screen, the space occupied by the toolbar becomes unavailable for display of other information or data. In a word processing system, the user may find that having more than two or three toolbars open at a time reduces the display screen area for the document being created or edited to an unacceptable extent. The loss of display screen area to toolbars is particularly noticeable if the user is running the display at a relatively low resolution, e.g., 640×480 pixels. Although the toolbars can be docked to one side or to the top or bottom of the screen so that they are not floating over and obscuring a document, the area that they occupy is not

available to display the text of the document. Consequently, the user is forced to choose between the convenience of readily accessing tools and applications by simply activating the corresponding graphic icons on the toolbars, or of having more display screen area available for text and other data. It would clearly be preferable to provide an alternative approach that enables the user to quickly access multiple toolbars without unduly limiting the screen area that is available for displaying text and graphics. Any such solution should allow the user the same kinds of flexibility in configuring toolbars and in positioning them at different points in the display as is presently available. The prior art does not appear to provide an acceptable solution to this problem.

SUMMARY OF THE INVENTION

In accordance with the present invention, a method is defined for providing access to a plurality of graphic objects in a computer display. The method includes the step of organizing the plurality of graphic objects into a plurality of generally quadrilaterally shaped toolbars. Each toolbar comprises a group of associated graphic objects organized in an array (one or two-dimensional). A stack of the plurality of toolbars is created on the computer display, so that any selected toolbar is fully visible and hides a substantial portion of any non-selected toolbar. (Note that the invention contemplates having more than one toolbar selected and fully visible.) A graphic object in any selected toolbar that is fully visible to a user on the computer display is directly selectable by the user to activate the graphic object. The user is enabled to choose any non-selected toolbar from among the plurality of toolbars, causing the toolbar that is thus chosen by the user to become a selected toolbar. The graphic objects on the chosen toolbar then become fully visible to the user on the computer display. A previously selected toolbar also then becomes a non-selected toolbar that is no longer fully visible, because a substantial portion of the previously selected toolbar is substantially hidden by the toolbar just chosen by the user.

Preferably, the graphic objects include buttons that are activated when the user clicks a select button on a pointing device while a cursor controlled by the pointing device is positioned over the button. To facilitate selection of a desired toolbar, each of the toolbars is provided with a characteristic identification that distinguishes that toolbar from at least some of the other toolbars disposed in the stack. The characteristic identification includes at least one alphanumeric character and/or logo that is disposed on the toolbar in a position remaining visible when a substantial portion of the toolbar is hidden by a selected toolbar.

A separate toolbar is added to the stack by enabling the user to select the separate toolbar with a pointing device and then to drag the separate toolbar onto the stack. Similarly, the method preferably further comprises the step of enabling the user to unstack the plurality of toolbars by selecting one of the toolbars comprising the stack and dragging that toolbar away from the stack. The toolbar dragged away becomes a separate toolbar that is no longer a part of the stack.

In addition, the method preferably includes the step of enabling the user to select the stack with a pointing device and to drag the stack to an edge of a window on the computer display screen, docking the stack at the edge. The user is also preferably enabled to change an orientation of the stack between vertical and horizontal, where the orientation relates to a longitudinal dimension of the plurality of toolbars comprising the stack.

The method also includes the step of causing one of the toolbars to slide in order to enable the toolbar chosen by the user to become fully visible. Preferably included are the steps of causing the toolbar that is sliding to decelerate as it approaches a rest position; and, causing the toolbar that is sliding to bounce before stopping. To enhance the realism of this sliding motion, it is associated with an audible sound.

In an auto-hide mode, the user is enabled to selectively hide the stack along an edge of a window on the computer display. In this mode, only a line of pixels comprising a border of the stack is visible at the edge of the window. Similarly, the user is enabled to selectively fully display the stack that is hidden in the auto-hide mode. Although unrelated, another feature enables the user to selectively autosize the stack to encompass a largest toolbar within the stack.

Yet another sizing feature of the method enables a user to selectively move the stack into a border region of a window on the computer display. In response to this action, the stack is caused to auto-fit within the border by adjusting dimensions of the stack and of the graphic objects that are fully displayed within any selected toolbar. The stack is positioned adjacent a window control in the border region.

The user is enabled to selectively float the stack on the computer display, and while the stack is floating, can modify a width and a length of the stack.

In another preferred step, the user is enabled to use a pointing device to select a graphic object appearing on the computer display outside of the stack. The graphic object that is selected can be dragged onto a toolbar comprising the stack, thereby adding the graphic object to the group of graphic objects within the toolbar. Optionally, the user can select one of the graphic objects comprising a toolbar and drag the graphic object to another toolbar for association with the group of graphic objects comprising the other toolbar. Alternatively, the user can select one of the graphic objects comprising a toolbar and drag the graphic object to a position outside of the stack, onto the computer display, causing the graphic object to become separated from the stack. The user is further enabled to select an object visible on the computer display with the pointing device; the object can then be dragged and dropped onto one of the graphic objects that is fully visible on any selected toolbar. In response, the graphic object is activated, and the object dropped serves as an input to an action that occurs as a result of the activation.

The user can select a plurality of properties for the stack.

A label identifying an object represented by each graphic object is displayed when the user moves a cursor over the graphic object. Furthermore, the label identifies a non-selected toolbar when the user moves the cursor over a visible portion of any non-selected toolbar.

Another aspect of the present invention is directed to a graphic operating system that is implemented on a computer. The graphic operating system includes graphic objects that appear on a computer display and comprises a plurality of means for implementing functions that are generally consistent with the steps of the method described above.

BRIEF DESCRIPTION OF THE DRAWING FIGURES

The foregoing aspects and many of the attendant advantages of this invention will become more readily appreciated as the same becomes better understood by reference to the following detailed description, when taken in conjunction with the accompanying drawings, wherein:

FIG. **1A** is a block diagram of a personal computer suitable for use in implementing the present invention;

FIG. **1B** is a block diagram showing some of principle components of the processor chassis in the personal computer of FIG. **1A**;

FIG. **2** (prior art) is a conventional horizontal "Office" toolbar on which are disposed a plurality of graphic object buttons that can be selected by the user to activate an object on a computer;

FIG. **3** is a horizontal stacked toolbar in accordance with the present invention, in which the Office toolbar of FIG. **1** is displayed as it appears stacked on top of a "Desktop" toolbar, typically docked at the top or bottom of the display screen;

FIG. **4** illustrates the stacked toolbar of FIG. **3** in which the Desktop toolbar has been selected to be on top of the Office toolbar;

FIG. **5** illustrates another stacked toolbar similar to that of FIG. **3**, but with a "Mouse" toolbar included in the stack;

FIG. **6** shows a "floating" stacked toolbar having the rectangular panel shape that is automatically assumed by the stacked toolbar of FIG. **4**, when undocked from the edge of the display screen;

FIG. **7** is a floating stacked toolbar in a rectangular panel configuration automatically assumed by the stacked toolbar of FIG. **3** when it is undocked from the edge of the screen;

FIG. **8** illustrates a vertical stacked toolbar similar to the horizontal stacked toolbar of FIG. **3**, as it appears when docked at the side of the display screen, and showing a "Tooltip" that identifies the function of a button on the toolbar;

FIG. **9** shows an upper portion of a vertical stacked toolbar, with a "Tooltip" displayed to identify an underlying toolbar;

FIG.**10** is a toolbar properties dialog box for the "View" tab;

FIG. **11** is a toolbar properties dialog box for the "Buttons" tab;

FIG. **12** is a toolbar properties dialog box for the "Toolbars" tab;

FIG. **13** is a graphic window showing a docked stacked toolbar and a stacked toolbar that has been autofit in the title region of the window;

FIG. **14** is a flow chart illustrating the steps of a user in selecting a hidden toolbar in a stack to be fully displayed;

FIG. **15** is a flow chart illustrating the logic implemented in handling the toolbars comprising a stacked toolbar;

FIG. **16** is a flow chart showing the logic steps in the autofit function used to position a stacked toolbar in the title bar;

FIG. **17** is a flow chart showing the steps implemented in resizing a stacked toolbar;

FIG. **18** is a Jackson structured programming diagram showing the steps involved in synchronously moving one of the toolbars from an initial position to its end position; and

FIG. **19** is state machine diagram illustrating an asynchronous method for sliding a toolbar from an initial position to an end position.

DESCRIPTION OF THE PREFERRED EMBODIMENT

With reference to FIG. **1A**, a generally conventional personal computer **10** is shown, which is suitable for imple-

menting the present invention. Personal computer **10** includes a processor chassis **12** in which are disposed a mother board and a plurality of other circuit boards (neither separately shown) of the type conventionally used in a personal computer. The mother board includes a central processing unit (CPU) **23** and memory **21**, including both read only memory (ROM) and random access memory (RAM) that are coupled to the CPU through a data bus **13**, as shown in FIG. **1B**. Also coupled to the CPU through the data bus are a display interface **15**, a hard drive and floppy drive interface **17**, a serial/mouse port **19**, and a keyboard interface **25**. Although many other internal components of processor chassis **12** are not shown, those of ordinary skill in the art will appreciate that such components and their interconnection are well known. Accordingly, further details concerning the internal construction of personal computer **10** need not be disclosed in connection with the present invention.

Personal computer **10** includes a floppy drive **14** and a hard drive **16** that are driven through the corresponding interface noted above, for use in writing and reading files stored on magnetic media. An operating system can thus be readily loaded on hard drive **16** by copying the necessary files from floppy disks that are inserted into floppy drive **14**. When the personal computer is "booted up" on the operating system from the files stored on hard drive **16**, the program instructions comprising these files are loaded into memory **21** and executed by CPU **23**, causing graphic elements of the operating system to be displayed to a user on a display **18**. The user interacts with programs executing on personal computer **10** by entering instructions or selections on a keyboard **20**, and/or through use of a mouse **22** (or other pointing device suitable for maneuvering a cursor on computer display **18**) to select and activate objects that are graphically displayed thereon.

The preferred embodiment of the present invention is designed to operate within a graphic operating system, such as Microsoft Corporation's WINDOWS operating system. However, it should be understood that the invention can readily be implemented in other graphic operating systems. A graphic operating system enables the user to select objects using mouse **22** or other suitable pointing device, thereby minimizing textual input required of the user on keyboard **20**. Typically, a graphic operating system provides graphic objects that can be activated when the user moves the cursor over a graphic object and selects it by clicking on the select button on the pointing device (typically, the left mouse button unless changed by the user). For example, if a graphic object represents a word processing program, double clicking on the graphic object causes the word processing program to be executed by the computer. To select a tool represented by a graphic object, the user moves the cursor over the graphic object and clicks once with the select button on the pointing device. As used in this description and in the claims that follow, a graphic object includes virtually any object that can be graphically displayed on a toolbar. In addition to graphic icons such as buttons that represent tools, hardware, programs, and controls, the preceding definition of graphic objects is intended to encompass, without limitation, other objects such as drop-down boxes, list boxes, and spinners.

FIG. **2** illustrates a typical toolbar on which a plurality of graphic objects are arranged in a linear array **28**. This example of a prior art toolbar, which comprises a plurality of buttons linearly arrayed in a line, side-by-side, has previously been used in such applications as Word for Windows **6** and Excel **5**, both distributed by Microsoft Corporation. Other applications have also used either a vertical or horizontal array of graphic objects that present various tools, controls, or programs that can be selected by the user with a pointing device.

As noted above in the Background of the Invention, the area available on a computer display for presenting a plurality of such toolbars or tool panels is limited, since each toolbar or array of graphic objects displayed on the screen takes up space that is needed for the text, data, and/or images of applications in which the user is working. To reduce the amount of space required to display a plurality of toolbars, some prior art designs have displayed a single toolbar and used tabs extending along the side of the toolbar that enable a user to selectively open other toolbars. Each tab is associated with a different toolbar or menu that is displayed when the tab is selected by a user. However, the tabs are not part of the toolbar that is already open. Furthermore, the tabs typically use space in the screen or window that would normally be available to an application. Examples of a tabbed toolbar appear in DELPHI™, a development program marketed by Borland Corporation.

As shown in FIGS. **3** through **5**, the present invention substantially reduces the space required to provide access to a plurality of different toolbars, by stacking the toolbars so that typically only the graphic objects associated with one or more of the toolbars in the are visible at a time. Although the embodiments shown in FIGS. **3** through **5** all contain only a single toolbar in which the graphic objects are fully visible, it is also contemplated that a plurality of toolbars in a stacked toolbar might be fully visible, for example, in a side-by-side configuration. A stacked toolbar **30** in FIG. **3** illustrates how a plurality of graphic objects including buttons **36***a*, **36***b*, **36***c*, and **36***d* are fully visible on the toolbar that is currently selected. This visible group of graphic objects is on a toolbar identified both with a text label **34** reading "Office" and by a distinctive logo **32**, which is also a control button. It will be apparent that the graphic objects visible on stacked toolbar **30** look similar to those on the prior art toolbar shown in FIG. **2**. However, stacked toolbar **30** differs substantially from the prior art toolbar, because a logo **42** (also a control button) is visible at one end of the stacked toolbar, but is not evident in the prior art toolbar. Logo **42** is associated with the graphic objects grouped on a "non-selected toolbar," i.e., a toolbar that is not fully visible. The graphic objects on the non-selected toolbar are on a portion of that toolbar that is hidden by the selected Office toolbar.

In a stacked toolbar **38** shown in FIG. **4**, the graphic objects associated with the toolbar identified by logo **42** and a text label **40** reading "Desktop" are fully visible. Note that in stacked toolbar **38**, the graphic objects associated with the Office toolbar are now hidden by the Desktop toolbar, so that only the logo **32** is displayed at one end.

Similarly, in FIG. **5**, a stacked toolbar **48** is shown in which the graphic objects comprising the Office toolbar are fully visible on a stack of toolbars that includes two other toolbars with groups of graphic objects. The first of these other toolbars is the Desktop toolbar. The logo associated with the Desktop toolbar appears at one end of the stack. A different group of graphic objects is disposed on a toolbar associated with a mouse logo **46** (also a control button), which is visible at the opposite end of the stack from the logo **42**. Although not specifically shown, the graphic objects associated with the Mouse toolbar are used for setting parameters that control the interactive operation of mouse **22**.

It should be noted that a logo and/or text may be assigned to a class of toolbars, rather than to a single toolbar.

Generally, a logo will be used with a toolbar or class of toolbars that logically represents the type of graphic objects included thereon. The border color of a toolbar is helpful in distinguishing one toolbar from another. A "tooltip" feature (discussed below) also helps to identify the various toolbars in a stack, in the event that a user is unfamiliar with or forgets the types of toolbars that are included in a stack.

In the preferred embodiment, the Z order (i.e., stacking order) remains constant as the selected (fully visible) toolbar is changed in response to a user's choice, to enable the graphic objects on a different toolbar to become visible. The toolbars that must move to reveal the graphic objects on a newly selected toolbar depends on the relative position of the newly selected toolbar in the stacking order. A simple example will illustrate the simplicity with which the stacked toolbars move to enable access of any graphic object on any toolbar in the stack. To access graphic objects **44***a* and **44***b* (FIG. **4**), when stacked toolbar **48** is on the display screen (FIG. **5**), the user simply moves the cursor with the pointing device so that the cursor rests over the visible portion of the partially hidden toolbar, e.g., over logo **42**, and then clicks the selector button on the pointing device. The previously selected Office toolbar will then slide left, exposing the group of graphic objects comprising the Desktop toolbar. Thereafter, if the user again wishes to access graphic objects **36***a* through **36***d* on the Office toolbar (FIG. **3** or FIG. **5**), the user selects logo **32** with the pointing device, causing any toolbar(s) hiding the graphic objects on the Office toolbar to slide away, so that graphic objects **36***a* through **36***d* (and all other graphic objects on the Office toolbar) are again displayed.

Stacked toolbars on which the graphic objects are arranged in a horizontally extending array, as shown in FIGS. **3** through **5**, are typically docked or anchored at the edge of the display screen or window so that the central portion of the display screen or window is available for any application that is then running on the computer. FIG. **13** shows a toolbar **48'** docked at the bottom of a window **70**. To dock a stacked toolbar, the user simply moves the stack until it is proximate the edge of the display screen or window. For example, as the stacked toolbar approaches the top or bottom edge, the stacked toolbar will automatically change to the single horizontal row array of graphic objects as shown in FIGS. **3** through **5** and anchor itself to the edge of the display screen or window as shown in FIG. **13**.

A stacked toolbar can also be docked on the left or right edges of the display screen or window to convert it to a vertical toolbar, such as a toolbar **30'**, which is shown in FIG. **8**. Toolbar **30'** includes the same graphic objects as toolbar **30** in FIG. **3**. Further, vertical stacked toolbar **30'** also includes a portion at its upper end on which is disposed logo **42**. When logo **42** is selected with the cursor and mouse **22** or other pointing device, the graphic objects of the Desktop toolbar, which corresponding to those in stacked toolbar **38** (in FIG. **4**), are displayed vertically. When not anchored at the edge of the display screen in a horizontal or vertical array of graphic objects, the stacked toolbars are described as floating, because they can be readily moved about.

When floating away from the screen or window edge, a stacked toolbars can be readily resized by the user. It is contemplated that a docked toolbar might also be resizeable; however, the preferred embodiment does not permit resizing of a docked toolbar. Although a floating stacked toolbar can also extend in a single row/column array of graphic objects if so sized by the user, a floating stacked toolbar is typically sized into a rectangular panel array of graphic objects (multiple columns and multiple rows of graphic objects), as shown in FIGS. **6** and **7**, and the rectangular panel can be oriented with its long dimension extending from left to fight or from top to bottom. The rectangular panel includes a title bar **54**, a minimize button **56** that minimizes the stacked toolbar to form a single graphic object on the display screen, and an option button **58** that is selected to display a menu of options or doubled clicked to close the stacked toolbar. To reshape a stacked toolbar when it is floating, the user simply moves the cursor until it is adjacent an edge of the toolbar. At this point, the cursor changes from a pointer into two parallel, spaced-apart lines representing a sizing tool, as is customary in the WINDOWS™ graphic operating system. By holding the select button on the pointing device and dragging, the side or end of the stacked toolbar selected by the sizing tool (cursor) is caused to move away from the respective opposite side or end, changing the shape of the stacked toolbar. Resizing is limited or controlled by a resizing algorithm to ensure that the area of the floating toolbar is sufficient to display all of the graphic objects on any toolbar in the stack. Thus, if the user decrease the length of the floating stacked toolbar, its width will increase automatically by an amount that accommodates the minimum area required by any toolbar in the stack.

A flowchart in FIG. **17** illustrates the steps involved in resizing a floating stacked toolbar, beginning with a block **320**. In a block **322**, the user drags an outer border of the stacked toolbar with the resizing tool by manipulating the mouse while the select button is depressed with cursor positioned on the border of the stacked toolbar. The system processes each of the toolbars in the stack when the resizing movement is complete, to determine how the stacked toolbar will be displayed. A decision block **324** determines if all of the graphic objects fit on a toolbar currently being processed. If not, a decision block **326** determines if the current toolbar needs to be taller (i.e., from top to bottom). If so, the logic in a block **328** increases the number of rows of graphic objects in the current toolbar so that all of the graphic objects will fit on the toolbar. The procedure then advances to a decision block **330**. However, if the toolbar does not need to be taller in decision block **326**, the logic proceeds to a block **332**, which decreases the number of rows of graphic objects, thereby making the current toolbar wider to accommodate all of the graphic objects on it. The logic thereafter proceeds to decision block **330**. Assuming that all graphic objects fit on the current toolbar being evaluated in decision block **324**, the logic simply drops to decision block **330** without changing the number of rows of graphic objects on the current toolbar.

In decision block **330**, the system determines if there are more toolbars to be processed, and if so, proceeds to a block **334**. At this point, the next toolbar in the stack is fetched for evaluation, returning to decision block **324**. Once all of the toolbars in the stack have been evaluated in this manner, the logic proceeds with a block **336** following decision block **330**.

Block **336** forms a union of all of the graphic objects on all of the toolbars in the stack. This step makes it possible to determine if changes that may have been made in the relative dimensions of the toolbars in blocks **328** and/or **332** will require a change in the layout of the stacked toolbar. Based on the requirements of the union formed of all of the graphic objects, a decision block **338** determines if the relative dimensions of the stacked toolbar must be changed. In other words, has the change made by the user resizing the stacked toolbar border caused the stacked toolbar to be wider than it is taller (compared to its relative orientation before resizing) or vice versa. If a change of this sort has occurred, a block

340 changes the orientation of the stacked toolbar accordingly. If no change is required, or after a change in orientation has been made, the logic continues with a block 342.

Block 342 sets the dimensions of all toolbars in the stack based upon the orientation and layout of the graphic objects on the toolbars following the resizing operation by the user. The stacked toolbar must accommodate the size requirements of the graphic objects on all of the toolbars, and thus will typically have an orientation and layout determined by the toolbar in the stack having the most graphic objects. A block 344 provides for setting the dimensions of the outer border of the stacked toolbar accordingly. The resizing procedure concludes in a block 346.

In FIG. 6, a stacked toolbar 50 is arranged as a panel on which are disposed graphic objects 44*a* and 44*b* (among others). A text label 52 and logo 42 appear at the top of the selected Desktop toolbar. Similarly, text label 34, which identifies the hidden toolbar panel on which the graphic objects comprising the Office group are disposed, is displayed at the left side, in a vertical orientation, with corresponding logo 32. By moving the cursor so that it appears within the portion of floating stacked toolbar 50 occupied by text label 34 and logo 32 and then depressing the select button on the pointing device, the user can cause the graphic objects comprising the Office group to be fully disclosed as shown on a stacked toolbar 60 in FIG. 7. The graphic objects comprising the Desktop group are then hidden by the Office toolbar and are not visible. To again selectively display the graphic objects in the Desktop toolbar, the user moves the cursor to the right end of stacked toolbar 60, so that it is on the portion in which text 52 and logo 42 are disposed, and then activates the select button on the pointing device. The graphic objects in the Desktop group are then fully revealed, as shown on stacked toolbar 50 in FIG. 6.

A floating stacked toolbar is moved by selecting its title bar 54 with the cursor and then moving the cursor with the select button on the pointing device held depressed. The stacked toolbar moves with the cursor and can be repositioned on the screen or window in the floating configuration or docked where desired along one of the edges. FIG. 13 illustrates another option in which a stacked toolbar 72 has been moved into the title bar of a window. An auto-fit algorithm sizes the stacked toolbar to fit within the title bar, adjusting the height of the graphic objects to equal the height of the title bar and strips away the border around the stacked toolbar. The auto-sized toolbar is docked in the title bar just to the left of the WINDOWS™ control buttons.

Referring to FIG. 8, a graphic object 37 that represents an object such as a file is shown to illustrate how it can be dragged about on the display screen using the mouse to select and move it and then dropped upon one of the displayed graphic objects within stacked toolbar 30'. In the example shown in this Figure, object 37 represents a document that the user wants to print. This document will be printed when it is dragged and dropped onto graphic object 36*b*, which corresponds to a printer (not shown) that is connected to the personal computer (or to a local area network to which the personal computer is coupled). When the select button is released to complete the drag-and-drop operation, the printer is activated, causing it to print the document represented by graphic object 37. Similarly, other files represented by graphic objects can be dragged and dropped onto a graphic object representing an application, thereby activating the application so that it runs using the file as an input, e.g., opening the document for editing in a word processing program. Files can also be dropped into folders.

A different type of drag-and-drop operation is used for adding graphic objects to a stacked toolbar. A graphic object that is dragged and dropped onto a toolbar in the stack while the ALT key is depressed is added to the group of graphic objects associated with that toolbar, at approximately the position where the graphic object is dropped. In addition, the graphic objects that are on the toolbar can be shifted left or right to create a space between existing graphic objects. If necessary, the stacked toolbar is resized to accommodate additional graphic objects, since the stacked toolbar has a size determined by the toolbar or group of graphic objects in the stack that requires the most space. A graphic object can also be moved from a toolbar and positioned in the window or display screen as a free floating graphic object. For example, in FIG. 13, graphic object 36*b*, which represents a printer, is shown as it is moved from the Office toolbar. It can be dropped in the position shown in the Figure, or moved onto the Desktop toolbar. A graphic object 74, representing a folder can then be dragged and dropped onto the Office toolbar to replace the printer.

Since a user may fail to recall the specific entity that is represented by a graphic object or by a logo of the toolbar in the stack, provision is made for displaying a text box label that identifies the graphic object or logo name, when the cursor is moved over the graphic object or logo. For example, in FIG. 9, a portion of a vertical stacked toolbar 110 is shown (similar to vertical stacked toolbar 30'). A cursor 112 is positioned so that it partly overlies logo 42, which is associated with the graphic objects comprising the Desktop toolbar. Accordingly, a text box 114 appears with the word "Desktop" displayed within it, identifying the toolbar represented by logo 42 for the user. A similar text box 115 is displayed in FIG. 8 as cursor 112 is moved over a graphic object on the displayed Office toolbar, indicating that the graphic object activates the AutoFormat function.

A further option provided for the stacked toolbars of FIGS. 3 through 5 is an auto-hide mode wherein the stacked toolbar is completely hidden except for a line a few pixels wide that extends vertically along the edge of the display screen. When the user jams the cursor against the edge of the display screen along which the fully hidden stacked toolbar is disposed, the stacked toolbar is again fully displayed, enabling the user to select any of the graphic objects on the selected toolbar or to choose a hidden toolbar so that its graphic objects are displayed. When the user moves the cursor off the fully displayed toolbar, it returns to the hidden position, i.e., appears to move off the screen or window. Optionally, the user can elect to have the fully hidden stacked toolbar "pop" into the fully displayed position, or can elect to have the hidden toolbar slide into the fully disclosed position, in an animated fashion. The auto-hide mode uses the least amount of display screen area, but the graphic objects on the selected toolbar are not available until the stacked toolbar is selectively displayed.

Various properties of a stacked toolbar can be selected in a properties dialog box 160 that is shown in FIGS. 10 through 12. In FIG. 10, a tab 162 is presently selected within dialog box 160, enabling the user to control the "View" properties of the stacked toolbar. A tab 163 labeled "Buttons" is selected in FIG. 11 to enable a user to configure the graphic objects used on a selected toolbar, and a tab 164 labeled "Toolbars" can be selected to enable a user to choose the specific toolbars that are included in a stack.

In the View properties, the user can select toolbar from a drop down box 165 and then set a Toolbar background color for the selected toolbar. The currently selected background color appears in a box 166. To change the color, the user selects a Change Color button 168, causing a screen with a color palette (not shown in the Figure) to be displayed, from

which the user can select the desired background color. A check box **170** also enables the user to indicate whether a gradient fill should be added to the text on the stacked toolbars. The term "gradient fill" refers to a color scheme in which the background color gradually varies from a dark to a lighter intensity in the area where the text label for the toolbar is disposed (for example, on the right side of floating stacked toolbars **50** and **60** in FIGS. **6** and **7**). A smooth characteristic for the background color is selected by placing a check in a check box **171**. A check box **172** enables the user to selectively indicate that a Standard Toolbar Color should be used. The Standard Toolbar Color is determined by defaults for the desktop set in the graphic operating system.

In an Options section **174**, a check box **176** allows the user to selectively indicate if large buttons are to be included on the stacked toolbars. Large buttons occupy more space but are easier to identify at higher screen resolutions, at which the graphics icons can be relatively small. A check box **178** provides for selectively indicating whether the Tooltips box (i.e., the label box identifying the graphic object or toolbar) is displayed when the user moves the cursor with the pointing device so that it is over one of the graphic objects or logos on the stacked toolbars.

In a check box **180**, the user can indicate if the stacked toolbars are always visible on top of the current window, whether docked or floating. Alternatively, the user can check a box **182** to enable the auto-hide mode for the stacked toolbars discussed above, so that only a line a few pixels wide appears along the edge of the screen when stacked toolbars are fully hidden. In a check box **184**, the user can elect to enable the auto fit option that enables toolbars to be docked in the title bar of the display screen or window. A check box **185** selectively enables animation of the toolbars so that one or more toolbars slide to disclose the graphic objects on a newly selected toolbar. Similarly, a check box **187** can be selected to enable sound to accompany the animation.

Two additional buttons are included within the dialog box, inside its bottom edge. An "OK" button **186** is the default that is highlighted, enabling the user to indicate that all of the toolbar properties have been set as desired. A Cancel button **188** closes the properties dialog box without changing any of the current properties that were previously selected by the user.

In FIG. **11**, a set of control **78** include a drop-down box **80** to enable the user to select a toolbar on which the buttons or other graphic objects are to be modified. In the example shown, an Accessories toolbar has been selected. The graphic objects within the Accessories toolbar are shown in a list box **82**. Any of the listed graphic objects can selectively be activated or deactivated to appear in the toolbar, as indicated by a check mark or absence of a check mark in a box adjacent the graphic object. "Move" control buttons **84** or **86** can be selected to shift any graphic object selected in list box **82** up or down within the list, thereby changing the relative position of the graphic object on the toolbar. Controls **88**, **90**, **92**, and **94** respectively enable a user to add a file, folder, or space, or delete a graphic object from the list displayed in list box **82**.

Similarly, in FIG. **12**, controls **100** are provided to enable a user to specify the toolbars that are to be included in the stack. A list of the toolbars are provided in a box **96**. Any of these listed toolbars can be activated for inclusion in the stack by selectively indicating the toolbar with a check mark in an adjacent check box. "Move" control buttons **84** and **86** are provided to change the relatively Z (stacking) order of the toolbars in the stack, by moving a selected toolbar up or down in the list. Controls **98** and **102** respectively enable a user to add a toolbar to the list or remove a selected toolbar from the list.

The steps implemented by a user to change the state of a stacked toolbar are shown in FIG. **14**, beginning at a start block **200**. In a decision block **210**, the user determines if a desired set of graphic objects in a toolbar are visible. If the toolbar desired by the user is visible, the user can select one of the graphic objects in the group of graphic objects associated with that toolbar. Accordingly, a block **216** indicates that the desired toolbar is revealed. Thereafter, the logic concludes in a stop block **218**. However, if the desired toolbar is hidden by a different selected toolbar, a block **212** indicates that the user scans the displayed text/logos of the other toolbars in the stack, to identify the desired toolbar. Thereafter, as noted in a block **214**, the user selects and clicks on the desired toolbar in the stack using the cursor and pointing device. That action causes the graphic objects on the newly selected toolbar that were previously hidden to be revealed, as noted in block **216**. The user can then select any of the graphic objects on the newly selected toolbar with the pointing device.

In FIG. **15**, the logic implemented by the user in changing the state of a stacked toolbar is shown as a series of steps. A decision block **220** determines if the stacked toolbar is in a desired orientation, and if so, proceeds to the result shown in a block **222**. At this point, no change in the orientation (or configuration) of the stacked toolbar is required. However, if the stacked toolbar is not in the desired orientation, a decision block **224** determines if the user wants to undock the stacked toolbar from the edge of the display screen or window, where it has previously been anchored. If so, the user selects the stacked toolbar with the cursor and pointing device and drags the stacked toolbar away from the edge of the screen or window. This step is indicated in a block **226**. A block **228** indicates that the stacked toolbar automatically forms a rectangular panel (a stacked toolbar like those shown in FIGS. **6** and **7**) as it is moved away from the edge of the display screen.

If the user is not undocking a stacked toolbar from the edge of the display screen, a decision block **230** determines if the user intends to dock a stacked toolbar that is floating. For an affirmative response, the logic flows to a block **232**, which indicates that the user has clicked the selector of the mouse or other pointing device while the cursor is on the title bar or background of the stacked toolbar. In a block **234**, the user drags the stacked toolbar, causing the stacked toolbar to automatically form a one-dimensional rectangle (single row or column of graphic objects) that snaps into a docked position at the adjacent edge of the display screen or window.

If the response to decision block **230** is negative, a decision block **236** determines if the user wants to resize a floating stacked toolbar. If so, a block **238** indicates that the user is clicking and dragging the border of the floating stack which has been selected after the cursor has changed to a resizing mode. In a block **240**, the user drags the border of the stacked toolbar to achieve the desired two-dimensional size and shape, subject to the autosizing limitations that ensure the area is sufficient to display the graphic objects included within any of the toolbars comprising the stacked toolbar.

A negative response to decision block **236** leads to a decision block **237**, which determines if the users wants to

auto-fit the stacked toolbar within the title bar at the top of the display screen or window. If so, a block **239** provides for enabling the auto-fit feature so that the use can click on the stacked toolbar title bar or background and then drag the stacked toolbar onto the title bar of the display screen or window. As the user drags the stacked toolbar onto the title bar, a block **241** indicates that the stacked toolbar becomes one-dimensional, forming a horizontal line of graphic objects in the title bar.

Details of the logic implemented in auto-fitting a stacked toolbar within the title bar are shown in FIG. **16**, beginning at a block **300**. In a block **302**, the logic sets the width of the stacked toolbar one-dimensional rectangle to fit the largest (i.e., the longest) toolbar in the stack. In a block **304**, the border of the stacked toolbar is stripped away. Next, in a block **306**, the graphic objects in the stacked toolbar are sized to equal the WINDOWS™ control button size, enabling them to fit within the title bar. A block **308** provides for docking the stacked toolbar at the top right corner of the window or display screen, right justified left of the WINDOWS™ controls in the title bar. The logic concludes in a block **310**.

With reference to FIG. **15**, a negative response to decision block **237** leads a decision block **242**. In this block, the graphic operating system determines if the user wants to remove a toolbar from the stack. If so, the user presses a designated control key while selecting the background of **20** while selecting the background of the top toolbar and then drags the selected toolbar with the pointing device to remove the selected toolbar from the stack, as indicated in a block **244**. As shown in a block **246**, the user drags the toolbar away from the stack to form a separate two-dimensional rectangle (multiple rows/columns) that is not stacked with any other toolbar.

A negative response to decision block **242** leads to a decision block **248**, which determines if the user wants to add an additional toolbar to the stack. If so, as indicated in a block **250**, the user does so by clicking and dragging the background of a floating toolbar that is not part of the stack, so that as indicated in a block **252**, the floating toolbar overlaps the stack. At this point, the toolbar that has been dragged into contact with the stacked toolbar snaps to the stack size and the graphic objects on the newly added toolbar are fully displayed. A negative response to decision block **248** returns to decision block **220**.

Following each of blocks **228**, **234**, **240**, **241**, **246**, and **252**, respectively, are nodes **254**, **256**, **258**, **259**, **260**, and **262**. These nodes represent the state of the stacked toolbar following the change caused by the actions of the user. Any further changes to the state of the stacked toolbar are made by returning to decision block **220** to repeat the logic just disclosed.

When a user selects a toolbar that is currently hidden to enable the graphic objects included therein to be fully displayed, the most direct technique to accomplish this task would be to simply replace the image of the graphic objects on the previously selected toolbar with the image of the graphic objects on the newly selected toolbar that was just chosen by the user, so that the graphic objects on the newly selected toolbar are fully displayed. The portion of the previously selected toolbar that includes the text and/or logo would then appear at one end of the stack after it is thus redrawn on the display screen. However, in the preferred embodiment of the present invention, a different approach was adopted for replacing the graphic objects previously selected toolbar with the graphic objects of a newly selected and previously hidden toolbar. Specifically, when the user selects a new toolbar by clicking on its logo, text, or on the exposed background area adjacent the logo or text, one or more toolbars appear to slide toward an end of the stack, exposing the graphic objects on the new selected toolbar. The sliding animation employed to expose the graphic objects on the newly selected toolbar is made even more realistic by providing an audible "swooshing" sound that accompanies the slide, and by causing any toolbar that is moving to decelerate as it approaches its final rest position and to appear to "bump" before settling to rest at that position.

There are two ways to implement the animated slide and bump used in the preferred embodiment. The first method is called a "synchronous slide", as indicated in a block **270** within FIG. **18**, because once the user initiates the sliding motion, it cannot be interrupted until the animation sequence is complete. The term "synchronous" is appropriate, because the animated sequence is performed as a singular operation that cannot change direction in mid-animation. The graphic operating system handles the apparent sliding motion, starting from an initial position in a block **272**. In carrying out the animated sliding motion that is referred to as a "SLIDE" in a block **274**, a value for the current position of the moving toolbar(s) is first determined in a block **282**. That position, which is represented by the variable POS', is equal to a function F_A(POS, END) in a block **282**, where END is the final position after the slide is completed, and POS is the current position of the moving toolbar(s). The animation is done by successively changing the displayed position of the moving toolbar(s) on the display screen each time that a predefined delay time has elapsed, as indicated in a block **286**, so that the position of the moving toolbar(s) is adjusted, as indicated in a block **284**. With rapid successive redraws of the display screen at a rate at least equal 15 times/second, the toolbar(s) appear to slide smoothly across the display screen.

F_A is an acceleration/deceleration function that modifies the apparent degree to which the position of the toolbar(s) changes after each successive predefined time delay. In a block **288**, a termination function F_C(POS, END, BUMP) controls the position of the sliding toolbar as it approaches the END position to implement the BUMP effect. A block **276** implements the apparent BUMP at the end of the slide by adjusting the position of the moving toolbar, as noted in a block **280**, after successive time delays, as indicated in a block **278**.

In the preferred embodiment, F_A determines the next position of the moving toolbar(s) by taking the previous position and adding the minimum of the constant "2" (pixels) and the function (END-POS/4). In accordance with the function F_C, the moving toolbar is shifted to a position, POS<END (or BUMP). In the preferred embodiment, the value of BUMP is a constant "4" (pixels) on the display screen.

A better way to handle the animation uses an asynchronous method, which is illustrated in the state machine diagram of FIG. **19**. If the asynchronous method is employed, the user is able to change the choice of a selected toolbar, before the graphic objects comprising the newly selected toolbar are fully exposed, and can instead select another toolbar. The asynchronous animation involves exposing the graphic objects on a newly selected toolbar and hiding the graphic objects on another toolbar that was previously selected. In the state machine shown in FIG. **19**, a change of state occurs with respect to the toolbar that includes the graphic objects to be displayed based upon

events caused by user input. The user input thus controls how the graphic objects on a toolbar are exposed. The states and events illustrated in state machine **290** are defined by the following Table 1.

TABLE 1

STATE	DEFINITION
Idle	Waiting for input or an event to occur
Prepare	Toolbar is readied to be moved
Slide	Toolbar is moved
Done	Movement of toolbar is completed
EVENT	DEFINITION
None	No event
New	New toolbar is selected
Same	Same toolbar is selected
Done	Toolbar reaches end position

The state machine starts in an initial IDLE state **292**. A time interval elapses between each change of state, and the changes are dependent upon the current input event. While in IDLE state **292**, state machine **290** takes no action if no user input occurs. If the user selects a new panel, the state changes to a PREPARE state in a block **294**. At this point, the user can select the originally selected toolbar (which corresponds to a NONE action), thereby returning to IDLE state **292**, or can select a NEW toolbar to be displayed, which leaves the state machine in the PREPARE state. If the user does not change the original selection, the state changes to SLIDE in a block **296**. At this point, the appropriate toolbar(s) starts sliding by incrementally changing position in subsequent screen redraws, in order to display the graphic objects on the newly selected toolbar. However, while the toolbar(s) are sliding to disclose the graphic objects, the user can select a NEW toolbar to be displayed, returning to the PREPARE state in block **294**, or can select the original toolbar to continue to be displayed (NONE), causing a return to the IDLE state, or can allow the slide to continue until it is DONE, as indicated in a block **298**. Once the slide is completed, the user can select yet another NEW toolbar to be displayed, returning to the PREPARE state in block **294** or can select no further action, leaving the state machine in the IDLE state in block **292**.

Although the present invention has been described in connection with the preferred form of practicing it, it will be understood by those of ordinary skill in the art that many modifications can be made thereto within the scope of the claims that follow. Accordingly, it is not intended that the scope of the invention in any way be limited by the above description, but that it be determined entirely by reference to the claims that follow.

The invention in which an exclusive right is claimed is defined by the following:

1. A method for providing access to a plurality of graphic objects on a computer display, comprising the steps of:

(a) organizing the plurality of graphic objects into a plurality of generally quadrilaterally shaped toolbars, each toolbar comprising a group of associated graphic objects organized in an array;

(b) creating a stack with the plurality of toolbars on the computer display, so that any selected toolbar is fully visible and hides a substantial portion of any non-selected toolbar from among the plurality of toolbars; a graphic object in any selected toolbar that is fully visible to a user on the computer display being directly selectable by the user to activate said graphic object; and

(c) enabling the user to choose any non-selected toolbar from among the plurality of toolbars of graphic objects, to make the non-selected toolbar that is thus chosen by the user become a selected toolbar that is fully visible to the user on the computer display, causing a previously selected toolbar to become a non-selected toolbar that is no longer fully visible, a substantial portion of the previously selected toolbar being substantially hidden by the toolbar just chosen by the user.

2. The method of claim **1**, wherein the graphic objects include buttons that are activated when the user clicks a select button on a pointing device while a cursor controlled by the pointing device is positioned over the button.

3. The method of claim **1**, wherein each of the toolbars is provided with a characteristic identification that distinguishes that toolbar from at least some of other toolbars disposed in the stack.

4. The method of claim **3**, wherein the characteristic identification includes at least one alphanumeric character that is disposed on the toolbar in a position so that said at least one alphanumeric character is visible when a substantial portion of the toolbar is hidden by a selected toolbar.

5. The method of claim **3**, wherein the characteristic identification comprises a logo that is disposed on the toolbar in a position so that the logo is visible when a substantial portion of the toolbar is hidden by a selected toolbar.

6. The method of claim **5**, wherein the logo includes at least one alphanumeric character.

7. The method of claim **1**, wherein a separate toolbar is added to the stack by enabling the user to select the separate toolbar with a pointing device and then to drag the separate toolbar onto the stack.

8. The method of claim **1**, further comprising the step of enabling the user to select the stack with a pointing device and to drag the stack to an edge of a window on the computer display screen, docking the stack at said edge.

9. The method of claim **1**, further comprising the step of enabling the user to change an orientation of the stack between vertical and horizontal, said orientation relating to a longitudinal dimension of the plurality of toolbars comprising the stack.

10. The method of claim **1**, further comprising the step of causing one of the toolbars to slide to a different position in order to enable the toolbar chosen by the user to become fully visible.

11. The method of claim **10**, further comprising the steps of causing said one of the toolbars that is sliding to decelerate as it approaches a rest position; and causing said toolbar that is sliding to bounce before stopping.

12. The method of claim **10**, further comprising the step of providing an audible sound that is associated with sliding said one of the toolbars.

13. The method of claim **1**, further comprising the step of enabling the user to unstack the plurality of toolbars, by selecting one of the toolbars comprising the stack and dragging said one toolbar away from the stack, causing said one toolbar to become a separate toolbar that is no longer a part of the stack.

14. The method of claim **1**, further comprising the steps of enabling the user to selectively hide the stack along an edge of a window in the computer display, in an auto-hide mode wherein only a line of pixels comprising a border of the stack is visible at the edge of the window; and, enabling the user to selectively fully display the stack that is hidden in the auto-hide mode.

15. The method of claim **1**, further comprising the step of autosizing the stack to encompass the group of graphic objects that is largest within the stack.

16. The method of claim 1, further comprising the steps of enabling a user to selectively move the stack into a border region of a window on the computer display; and, in response, causing the stack to auto-fit within the border by adjusting dimensions of the stack and of the graphic objects fully displayed within any selected toolbar, said stack being positioned adjacent a window control in the border region.

17. The method of claim 1, further comprising the step of enabling the user to selectively float the stack on the computer display, and while the stack is floating, enabling the user to modify a width and a length of the stack.

18. The method of claim 1, further comprising the step of enabling the user to use a pointing device to select a graphic object appearing on the computer display outside of the stack; and, to drag the graphic object that is selected onto a toolbar comprising the stack, thereby adding the graphic object to the group of graphic objects within the toolbar.

19. The method of claim 1, further comprising the step of enabling the user to select one of the graphic objects comprising a toolbar and to drag said one graphic object to another toolbar for association with the group of graphic objects contained therein.

20. The method of claim 19, further comprising the step of enabling the user to select one of the graphic objects comprising a toolbar and to drag said one graphic object to a position outside of the stack on the computer display, causing said one graphic object to become separated from the stack.

21. The method of claim 1, further comprising the step of enabling the user to select an object visible on the computer display with a pointing device; and, to drag and drop the object onto one of the graphic objects comprising the selected toolbar that is fully visible, thereby activating said one of the graphic objects and serving as an input to an action that occurs when said one of the graphic objects is activated.

22. The method of claim 1, further comprising the step of enabling the user to select a plurality of properties for the stack.

23. The method of claim 1, further comprising the step of displaying a label identifying an object represented by each graphic object when the user moves a cursor over the graphic object.

24. The method of claim 1, further comprising the step of displaying a label identifying a non-selected toolbar when the user moves the cursor over a visible portion of any non-selected toolbar.

25. A graphic operating system that is implemented on a computer, said graphic operating system including graphic objects that appear on a computer display, comprising:

(a) means for organizing the plurality of graphic objects into a plurality of generally quadrilaterally shaped toolbars, each toolbar comprising a group of associated graphic objects organized in an array;

(b) means for creating a stack of the toolbars on the computer display so that any selected toolbar substantially hides a substantial portion of any non-selected toolbar in the stack; said group of graphic objects in any selected toolbar being fully visible on the computer display to a user so that a graphic object within said group is directly selectable and activatable by the user; and

(c) means for enabling the user to choose a non-selected toolbar to become a selected toolbar, including means for causing the non-selected toolbar thus chosen to become fully visible so that the graphic objects comprising it are visible to the user on the computer display, and so that a substantial portion of a previously selected toolbar is hidden by the toolbar that was just chosen by the user, said graphic operating system thereby reducing an area of the computer display required for displaying the groups of graphic objects comprising the toolbars in the stack.

26. The graphic operating system of claim 25, further comprising a central processing unit that is coupled to the computer display, and memory for storing a plurality of program instructions, wherein the means for organizing, means for creating, and means for enabling are effected on the computer by executing the plurality of program instructions with the central processing unit.

27. The graphic operating system of claim 25, further comprising docking means for enabling the user to select the stack and position it at an edge of a window on the computer display where it remains docked.

28. The graphic operating system of claim 25, further comprising sizing means that enable the user to graphically alter dimensions of the stack.

29. The graphic operating system of claim 25, wherein the sizing means further enable the user to float the stack, and to change its relative horizontal and vertical dimensions while it is floating.

30. The graphic operating system of claim 25, further comprising means for animating one of the toolbars to enable the toolbar chosen by the user to become fully visible, so that said one of the toolbars slides across the stack and decelerates to a stop, with an accompanying audible sound.

31. The graphic operating system of claim 25, further comprising means for adding and removing a selected graphic object respectively to and from the groups of graphic objects comprising the toolbars in the stack, by dragging and dropping the selected graphic object.

32. The graphic operating system of claim 25, further comprising means for adding and removing a specific toolbar respectively to and from the plurality of toolbars in the stack, by dragging and dropping the specific toolbar.

33. The graphic operating system of claim 25, further comprising means for autosizing the stack to accommodate a largest of the plurality of toolbar comprising the stack.

* * * * *

Exhibit D

US006263352B1

(12) **United States Patent**
Cohen

(10) **Patent No.: US 6,263,352 B1**
(45) **Date of Patent: Jul. 17, 2001**

(54) **AUTOMATED WEB SITE CREATION USING TEMPLATE DRIVEN GENERATION OF ACTIVE SERVER PAGE APPLICATIONS**

(75) Inventor: **Michael A. Cohen**, Seattle, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/970,217**

(22) Filed: **Nov. 14, 1997**

(51) **Int. Cl.**[7] **G06F 17/21**; G06F 15/16
(52) **U.S. Cl.** **707/513**; 707/501; 709/203
(58) **Field of Search** 707/500–531; 709/200–232; 705/1–5, 26; 706/45–47; 345/326–338

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,035,119 * 3/2000 Massena et al. 717/1
6,055,541 * 4/2000 Solecki et al. 707/103

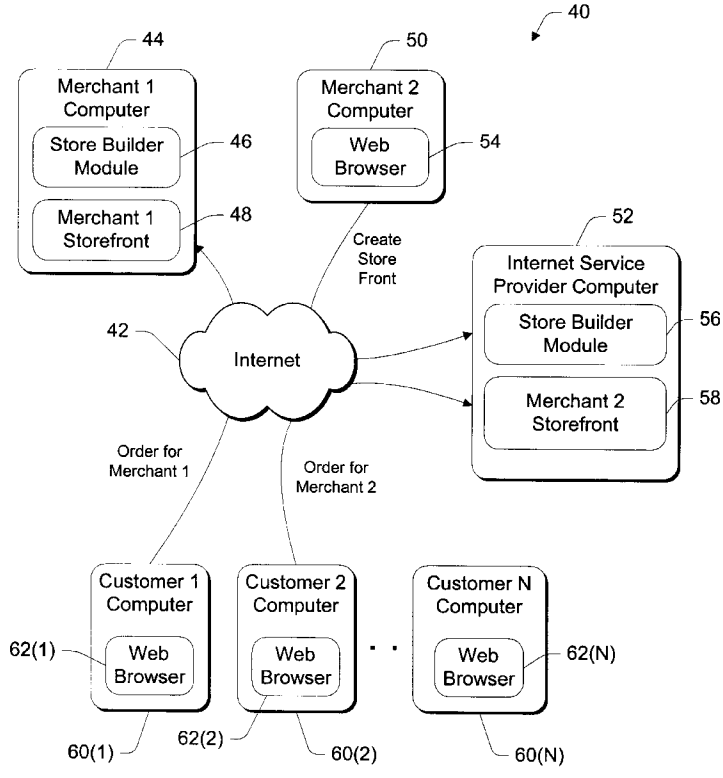
* cited by examiner

Primary Examiner—Hosain T. Alam
Assistant Examiner—Alford W. Kindred
(74) *Attorney, Agent, or Firm*—Lee & Hayes, PLLC

(57) **ABSTRACT**

A computer-implemented system is designed to assist a merchant in setting up an electronic online storefront that is customized to the merchant's business, without requiring the merchant to program. The system employs a store builder wizard to guide a merchant through a series of questionnaires designed to extract information pertaining to the merchant's business. The system further employs a page generator to create active server pages (ASPs) that form the customized storefront. The page generator creates the active server pages from a set of templates that are generic to formation of an online storefront. The templates are written as an extension to the active server page technology in a combination of hypertext language and scripting language. The active server templates specify an additional, higher order scripting level that distinguishes a second level of code by a new delimiter. During creation of the storefront, the page generator reads each active server template file and converts it to a scripting program having executable lines of code derived from the higher-order level of instructions denoted by the new delimiters. The page generator then executes the scripting program using the merchant data as input to produce a customized set of active server pages. The resulting active server pages contain the hypertext language and the lower-order level of instructions in the scripting language denoted by the original delimiters. The active server pages are stored together to form an active server application customized to the merchant's storefront.

17 Claims, 7 Drawing Sheets



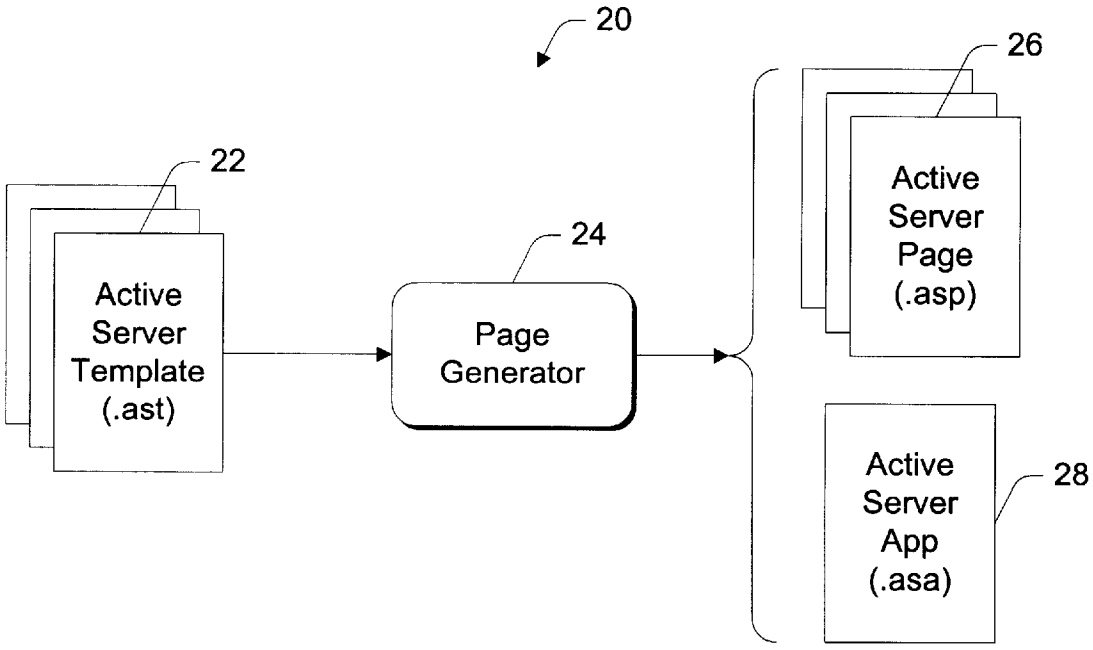

20
22
Active Server Template (.ast)
24
Page Generator
26
Active Server Page (.asp)
Active Server App (.asa)
28

Fig. 1

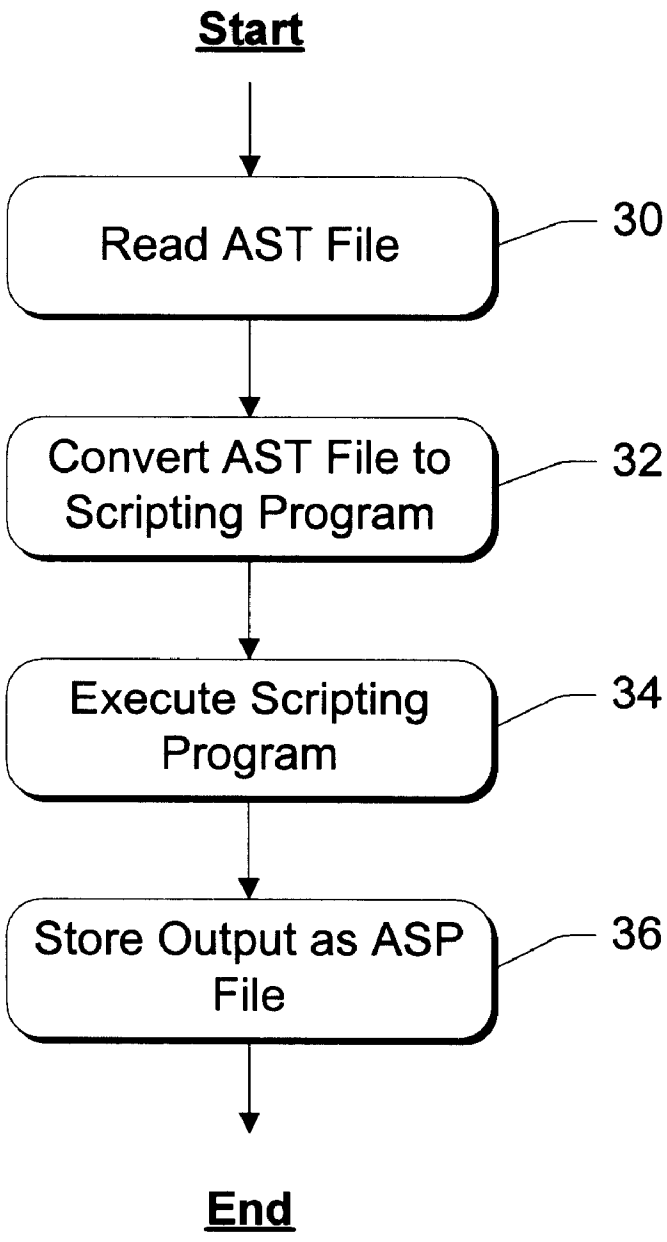

Start
Read AST File
30
Convert AST File to Scripting Program
32
Execute Scripting Program
34
Store Output as ASP File
36
End

Fig. 2

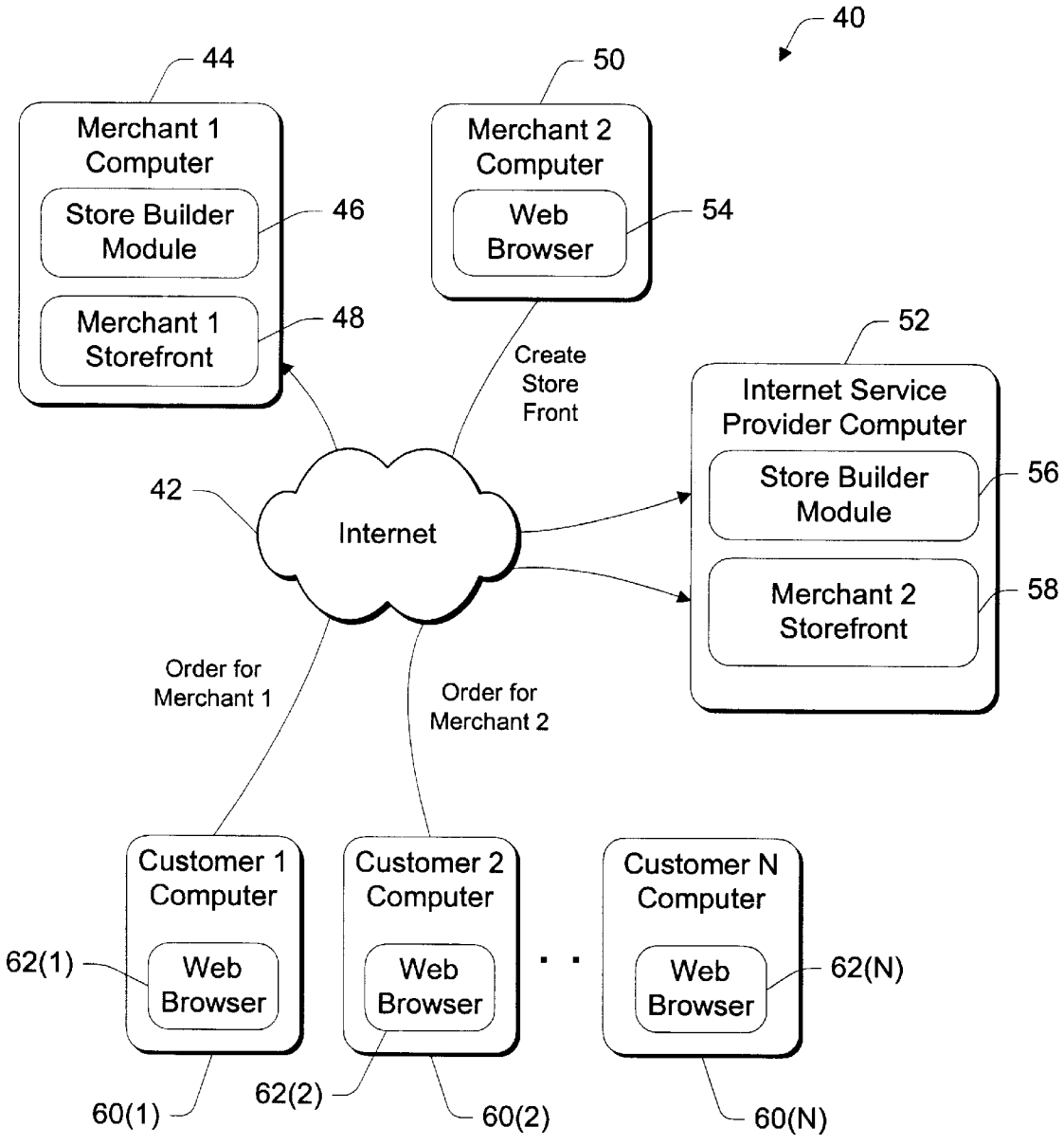

40
44
Merchant 1 Computer
Store Builder Module
46
Merchant 1 Storefront
48
50
Merchant 2 Computer
Web Browser
54
Create Store Front
52
Internet Service Provider Computer
Store Builder Module
56
Merchant 2 Storefront
58
42
Internet
Order for Merchant 1
Order for Merchant 2
Customer 1 Computer
Web Browser
62(1)
60(1)
Customer 2 Computer
Web Browser
62(2)
60(2)
Customer N Computer
Web Browser
62(N)
60(N)

Fig. 3

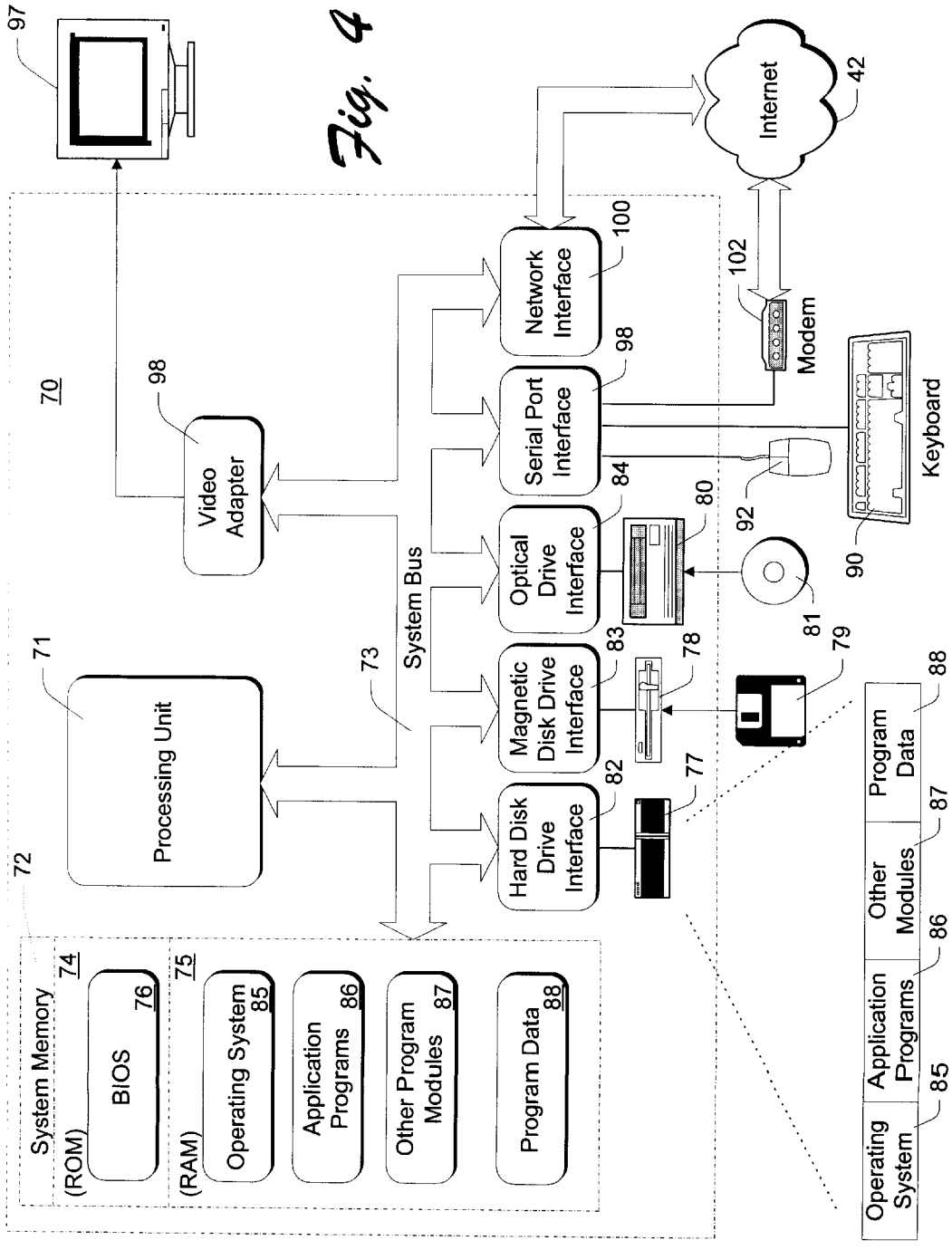


Fig. 4
97
Internet
42
70
98
Video Adapter
Network Interface
100
102
Modem
Serial Port Interface
98
Keyboard
Optical Drive Interface
84
80
92
90
81
71
Processing Unit
73
System Bus
Magnetic Disk Drive Interface
83
78
79
72
Hard Disk Drive Interface
82
77
System Memory
74
(ROM)
BIOS
76
75
(RAM)
Operating System
85
Application Programs
86
Other Program Modules
87
Program Data
88
Program Data
88
Other Modules
87
Application Programs
86
Operating System
85

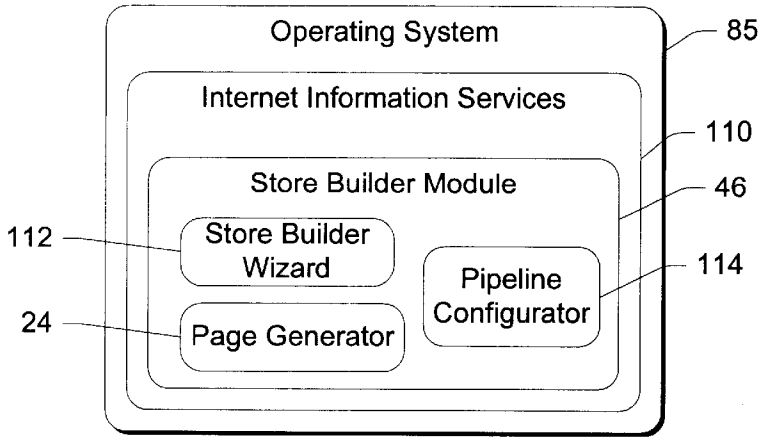

Operating System
85
Internet Information Services
110
Store Builder Module
46
112
Store Builder Wizard
Pipeline Configurator
114
24
Page Generator

Fig. 5

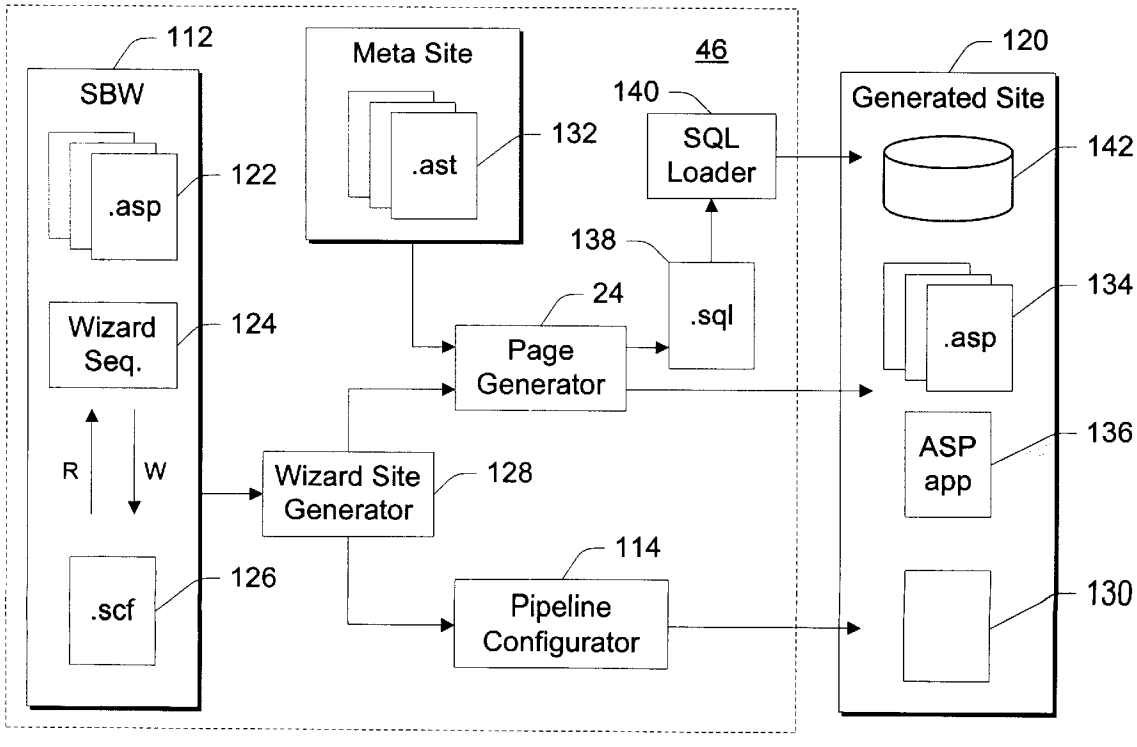

112
SBW
.asp
122
Wizard Seq.
124
R
W
.scf
126
Meta Site
.ast
132
46
140
SQL Loader
138
.sql
24
Page Generator
Wizard Site Generator
128
114
Pipeline Configurator
120
Generated Site
142
.asp
134
ASP app
136
130

Fig. 6

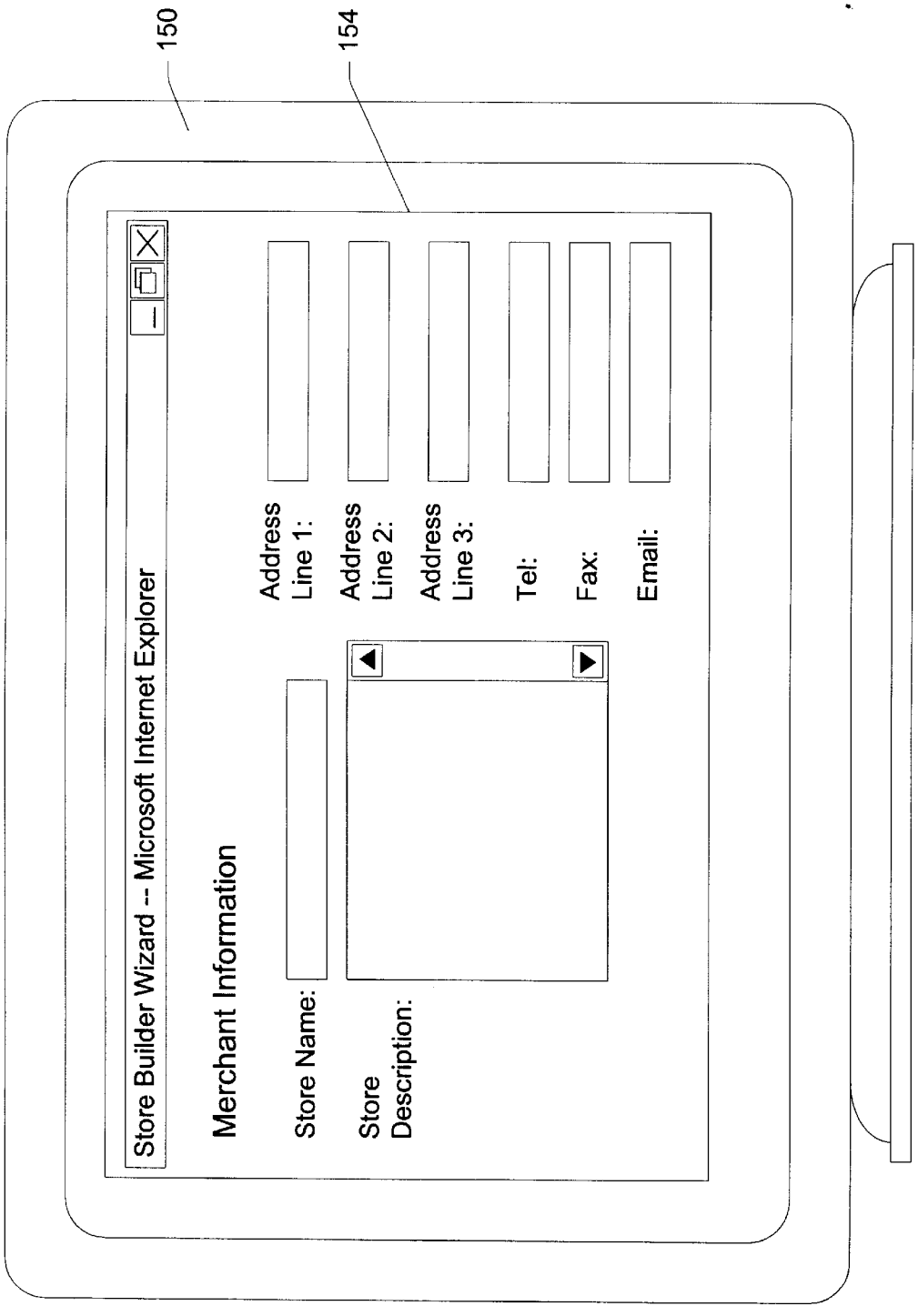

150
154
Store Builder Wizard -- Microsoft Internet Explorer
Merchant Information
Store Name:
Store Description:
Address Line 1:
Address Line 2:
Address Line 3:
Tel:
Fax:
Email:

Fig. 7

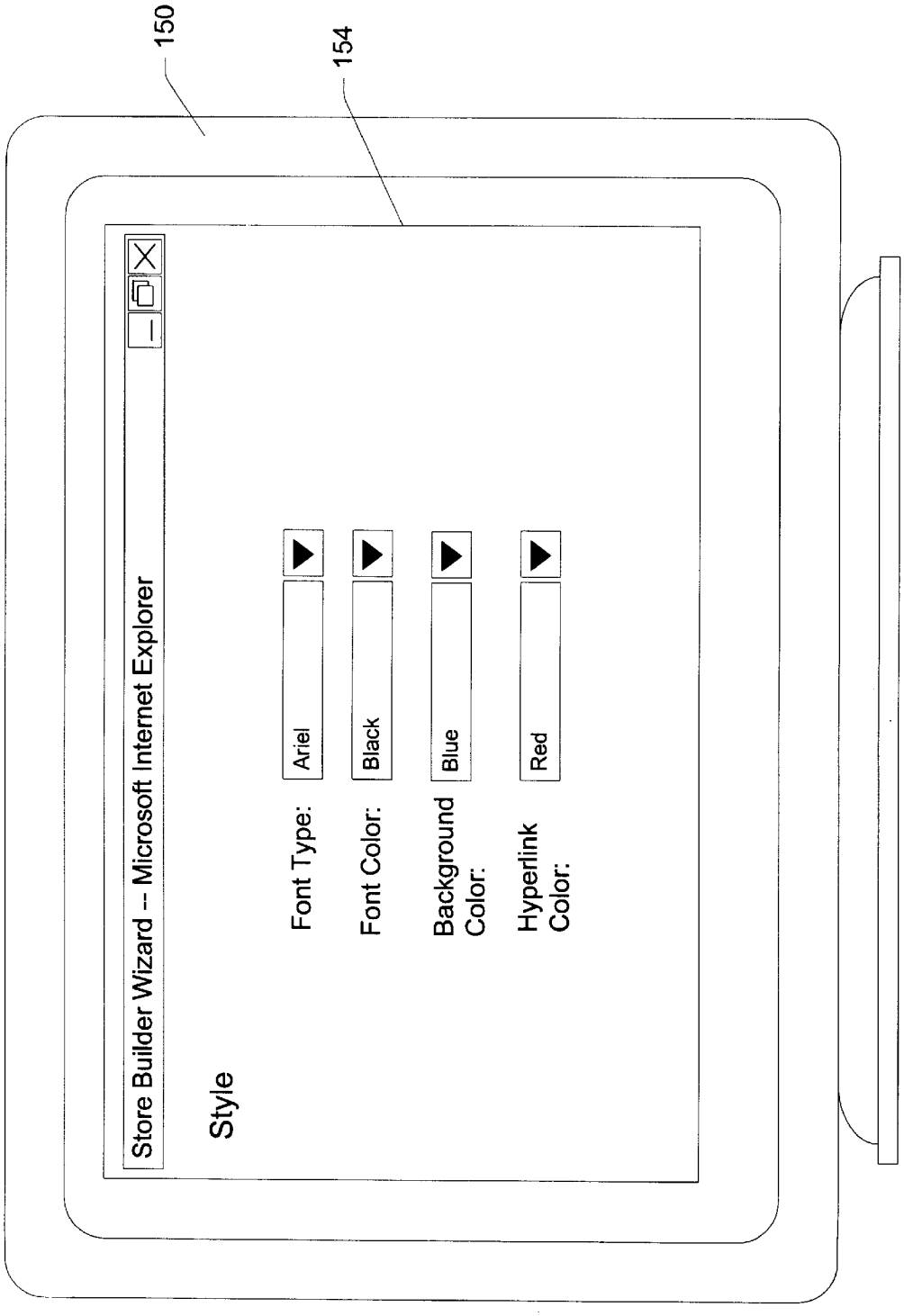

150
154
Store Builder Wizard -- Microsoft Internet Explorer
Style
Font Type:
Ariel
Font Color:
Black
Background Color:
Blue
Hyperlink Color:
Red

Fig. 8

AUTOMATED WEB SITE CREATION USING TEMPLATE DRIVEN GENERATION OF ACTIVE SERVER PAGE APPLICATIONS

TECHNICAL FIELD

This invention relates to computer network servers, such as Internet servers. More particularly, the invention relates to sever operating systems that enable generation of active server page applications, which are used to support Web sites on the Internet, from active server templates.

BACKGROUND

Online commerce is experiencing dramatic growth in recent years. Merchants are developing sites on the World Wide Web (or simply "WWW" or "Web") at a rapid pace. With Web sites in place, consumers can access and order goods and/or services electronically over the Internet from the comfort of their own homes or offices. It is becoming fairly common for a consumer to browse a merchant's catalog online, select a product, place an order for the product, and pay for the product all electronically over the Internet.

Merchants want Web sites that are customized to their product line. Ideally, merchants might like to design their own Web site to create a desired shopping atmosphere suitable for their products and services. Unfortunately, most merchants do not have the technical expertise to create and maintain a Web site on the Internet. As a result, merchants typically hire independent consulting firms to create and/or manage Web sites on the merchants' behalf

It would therefore be beneficial to design a system that permits a merchant to create its own Web site without requiring the merchant to possess software design and programming skills.

To aid the following discussion, it might prove useful to provide additional background information on how resources are formatted and rendered over the Internet. Resources available on the Internet are most commonly presented as hypertext. "Hypertext," also referred to as "hypermedia," is a metaphor for presenting information in which text, images, sounds, and actions become linked together in a complex, non-sequential Web of associations that permit a user to browse through related topics, regardless of the presented order of the topics. Hypertext content is widely used for navigation and information dissemination on the Web. A "Web browser" is normally used to retrieve and render hypertext content from the Web.

Hypertext content is commonly organized as documents with embedded control information. The embedded control information includes formatting specifications, indicating how a document is to be rendered by the Web browser. In addition, such control information can include links or "hyperlinks," which are symbols or instructions telling the Web browser where to find other related VWeb documents on the Internet.

Hypertext content is commonly written in a "markup language." "SGML" (Standard Generalized Markup Language) is one such hypertext language, defined formally as "a language for document representation that formalizes markup and frees it of system and processing dependencies." SGML is a language for describing the structure of documents and for describing a tagging scheme to delineate that structure within text.

For creating hypertext content, Web documents utilize a subset of SGML called "HTML" (Hypertext Markup Language). An HTML textual document can be thought of as plain text that contains formatting instructions in the form of HTML markup codes or "tags." Tags tell Web browsers how to render and print documents, and are also used to specify hyperlinks.

The following is a simple example of a portion of an HTML document containing a single hyperlink:

Microsoft has a Web page with the latest <A HREF= "HTTP://www.microsoft.com/upgrades"> upgrades</A> to its popular word processing program.

The angled brackets define hypertext tags. When rendered by a Web browser, the word "upgrades" would appear highlighted and/or underlined to the user, and the text within the angled brackets would not appear at all, as follows:

Microsoft has a Web page with the latest upgrades to its popular word processing program.

By clicking on the highlighted keyword "upgrades," the user can instruct the Web browser to activate the underlying URL. In this case, the underlying URL is to an HTTP (hypertext) document located at host computer "www.microsoft.com," having the file name "upgrades."

Hypertext usage is not limited to the Internet. Various multimedia applications utilize hypertext to allow users to navigate through different pieces of information content. For instance, an encyclopedia program might use hyperlinks to provide cross-references to related articles within an electronic encyclopedia. The same program might also use hyperlinks to specify remote information resources such as Web documents located on different computers.

Microsoft Corporation has recently introduced a technology referred to as "Active Server Pages." An active server page, or "ASP", allows a user to write Web pages using a combination of a hypertext language (e.g., HTML) and a scripting language, such as Visual Basic from Microsoft Corporation or Java™ from Sun Microsystems. As an example, the following ASP file contains scripting language to define the colors used in the web page for the background, hyperlinks, and text.

```
<HTML>
<BODY
   bgcolor=<%=Application("color_bgcolor")%>
   link=<%=Application ("color_link")%>
   text=<%=Application(("color_text")%>
>
<P>Colored text here.
</BODY>
</HTML>
```

The hypertext terms are set apart by the angled brackets "<" and ">" such as "<HTML>" and "<BODY>". The delimiters "<%" and "%>" denote the instructions in the scripting language. When the ASP file is read and rendered by a Web browser, the scripting instructions within the delimiters are executed to fill in the background color, link color, and text color. The result is a familiar hypertext document.

Active Server Pages are described in documentation available from Microsoft's Web site "www.microsoft.com", under the section Internet Information Services. This text is hereby incorporated by reference.

SUMMARY

This invention provides a computer-implemented system that enables a user to create a web site that is customized to the user's needs without requiring the user to program. As one particular example, the system is configured to assist a

merchant in setting up an electronic online storefront that is customized to the merchant's business.

According to one aspect of this invention, the system employs a store builder wizard to guide a merchant through a series of questionnaires designed to extract information pertaining to the merchant's business. For example, the questionnaires might be written as a series of HTML documents that require the merchant to enter data concerning the business' address, inventory, pricing, preferred method of payment, and so forth. The answers to the questions are stored in a data file.

The system further employs a page generator to create active server pages (ASPs) that form the customized storefront. The page generator creates the active server pages from a set of templates that are generic to formation of online storefronts. The page generator uses the merchant data collected by the store builder wizard as input to the templates to thereby convert the templates to ASPs that are customized according to the merchant's input. The active server pages are stored together to form an active server application that supports the merchant's storefront. In this manner, the merchant merely enters data through a user-friendly wizard interface and a customized storefront is automatically created. The merchant is not required to have any programming skills.

According to another aspect of this invention, the templates are written as an extension to the active server page technology. The templates, which are referred to as "active server templates", are written in a combination of hypertext language and scripting language. The active server templates are thereby akin to active server pages. However, unlike ASPs, the active server templates specify an additional, higher order scripting level that distinguishes a second level of code by a new delimiter.

During creation of the storefront, the page generator reads an active server template file and converts it to a scripting program having executable lines of code derived from the higher-order level of instructions denoted by the new delimiters. The page generator then executes the scripting program using the merchant data as input to produce a customized active server page. The resulting active server page contains the hypertext language and the lower-order level of instructions in the scripting language denoted by the original delimiters.

In the described implementation, the store builder wizard and page generator are embodied in a server operating system that executes on a network server. The store builder wizard and page generator can be accessed remotely over the Internet using a Web browser. In this arrangement, the merchant can access the host network server and enter the merchant data. The host server creates a custom storefront from the merchant data, and thereafter manages the storefront on behalf of the merchant.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a diagrammatic illustration of a computer-implemented Web page generation system.

FIG. **2** is a flow diagram having steps in a method for transforming active server templates into active server pages.

FIG. **3** shows an online commerce system as one possible environment in which the page generation system might be used to generate merchant storefronts.

FIG. **4** shows a host computer that can be configured to implement the page generation system.

FIG. **5** is a block diagram showing the page generation system implemented in software as part of a server operating system on the host computer of FIG. **4**.

FIG. **6** shows a system software architecture for a store builder module that implements the page generation system used to build merchant storefronts.

FIGS. **7** and **8** show examples of two wizard pages that guide a merchant through a series of questions to collect data pertaining to the merchant's business.

DETAILED DESCRIPTION

Web sites on the Internet are commonly formed from a set of one or more Web pages. Individual Web pages are typically configured as a hypertext document, such as an HTML document. As noted in the Background, a recent technology enables a Web page to be configured as an "active server page", or "ASP". An active server page is written in a combination of a hypertext language (e.g., HTML) and a scripting language, such as Visual Basic Script (or "VBS") or JScript from Microsoft Corporation, per, python, REXX, or tcl. When a browser requests an ASP, the scripting language is executed to produce a Web page in the form of a hypertext document that can be rendered by the browser.

One aspect of this invention concerns creation of Web pages that may be used in a Web site. More particularly, one inventive aspect pertains to a computer-implemented system that enables a user to input information pertaining to a Web site and then automatically generates a set of Web pages based upon the information. In the context of an online business, the system permits a merchant to create its own customized online storefront simply by inputting data pertaining to its business. The merchant needs no special programming skills to build the storefront.

At the heart of the site creating system is a page generation system that enables automatic production of Web pages using site-relevant data as input. The page generation system generates custom Web pages from templates that are generic to a variety of Web sites.

General Page Generation System

FIG. **1** shows a computer-implemented page generation system **20** that generates Web pages from templates. In the preferred implementation, the Web pages are written as active server pages. The page generation system **20** includes a library of templates **22** and a page generator **24** that converts one or more of the active server templates **22** into multiple active server pages **26** and an active server application **28**. Together, the active server pages **26** (designated as ".asp") and the active server application **28** (designated as ".asa") form an ASP application, wherein each ASP application consists of a single global active server application **28** and multiple active server pages **26**.

The templates **22** are written in a combination of hypertext language (e.g., HTML) and scripting language (e.g., VBS or JScript), which are the same two languages used in active server pages. The templates are referred to as "active server templates" or "ASTs". Unlike ASPs, however, the active server templates specify an additional, higher order scripting level that contain instructions denoted by a new delimiter pair "<%%" and "%%>". The new delimiters distinguish the higher order scripting level from the original, lower order scripting level that is denoted by the delimiter pair "<%" and "%>". Due to the different delimiters, the higher order scripting level can be executed independently of the lower order scripting level. Consider the following AST file:

```
<HTML>
<BODY
<%%if Item.SpecifyColors then%%>
  bgcolor=<%=Application("color_bgcolor")%>
  link=<%=Application("color_link")%>
  text=<%=Application("color_text")%>
<%% end if %%>
>
<P>Colored text here.
</BODY>
</HTML>
```

The AST file resembles the example ASP file described in the Background, except that it contains an additional level of scripting code in the form of an "if statement" set apart by the delimiters "<%%" and "%%>". The "if statement" can be executed separate from the original scripting code pertaining to color selection of the background, hyperlinks, and text, as denoted by the delimiters "<%" and "%>".

FIG. **2** shows a method implemented in the page generator **24** to transform the active server template **22** into a corresponding active server page **26**. At step **30**, the page generator **24** reads the AST file from memory. The page generator **24** then converts the AST file to a scripting program (step **32**). As one example, the page generator **24** turns every line in the AST file into a print statement, except the higher order lines of code surrounded by the "<%%" and "%%>" delimiters. The following scripting program is produced:

```
Print "<HTML>"
Print "<BODY"
if Item.SpecifyColors then
Print
  "bgcolor=<%=Application("color_bgcolor")%>"
Print "link=<%=Application("color_link")%>"
Print "text=<%=Application("color_text")%>"
end if
Print ">"
Print "<P>Colored text here."
Print "</BODY>"
Print "</HTML>"
```

At step **34**, the page generator **24** executes the scripting program. The property "Item.SpecifyColors" is a data item that is input to the program. In the context of Web site generation, this property is obtained from the user. Assuming the property "Item.SpecifyColors" is true, executing the scripting program would yield the following output:

```
<HTML>
<BODY
  bgcolor=<%=Application("color_bgcolor")%>
  link=<%=Application("color_link")%>
  text=<%=Application("color_text")%>
>
<P>Colored text here.
</BODY>
</HTML>
```

This output is stored as an ASP file (step **36** in FIG. **2**). The ASP file can later be retrieved and rendered to HTML by executing the lines of code denoted the "<%" and "%>" delimiters. The resulting HTML is then passed to a browser, which renders the HTML on the screen.

Exemplary Environment

For purpose of continuing discussion, the page generation system is described within an exemplary environment of online commerce. In this environment, a merchant desires to create an online storefront that is customized to its business. It is noted, however, that the page generation system of FIGS. **1** and **2** can be used in contexts other than the online commerce environment.

FIG. **3** shows an online commerce system **40** in which customers shop for goods and services from merchants over the Internet **42**. The online commerce system **40** exhibits two possible scenarios. One scenario is that the merchant creates and manages its own online storefront. Merchant **1** represents this case. A host computer **44**, which resides at merchant **1**, is loaded with a store builder module **46** that aids the merchant in creating an online storefront. The merchant enters data relevant to its business and the store builder module **46** generates a storefront **48** that is customized to the merchant's business based on the entered data. The merchant storefront **48** is kept and managed at the merchant's computer **44**.

The second scenario is where a merchant has no expertise in managing an online storefront. Hence, the merchant relies on the expertise of an Internet Service Provider (ISP). Merchant computer **50** and ISP computer **52** represent this case. The ISP computer **52** is configured with the store builder module **56**. The merchant uses a local Web browser **54** to remotely access the store builder module **56** on the ISP computer **56** to enter data pertaining to the merchant's business. The store builder module **56** creates a merchant's storefront **58** based on the data submitted by the merchant over the Internet **42**. The storefront **58** is maintained at the ISP computer **52**.

The customers access the storefronts electronically over the Internet **42** from their computers, as represented by customer computers **60(1)**, **60(2)**, . . . , **60**(N). Each customer computer is configured with a Web browser **62(1)**, **62(2)**, . . . , **62**(N). To shop and order goods from merchant **1**, a customer (e.g., customer **1**) uses his/her Web browser to access the merchant storefront **48** maintained on the merchant host computer **44**. To shop and order goods from merchant **2**, a customer (e.g., customer **2**) uses the Web browser to access the merchant storefront **58** maintained on the ISP host computer **52**.

FIG. **4** shows an example implementation of a host computer, such as the merchant host computer **44** or the ISP host computer **52**. The host computer is a general purpose computing device in the form of a conventional personal computer **70** that is configured to operate as a host network server. The server computer **70** includes a processing unit **71**, a system memory **72**, and a system bus **73** that couples various system components including the system memory **72** to the processing unit **71**. The system bus **73** may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory **72** includes read only memory (ROM) **74** and random access memory (RAM) **75**. A basic input/output system **76** (BIOS) is stored in ROM **74**.

The server computer **70** also has one or more of the following drives: a hard disk drive **77** for reading from and writing to a hard disk, a magnetic disk drive **78** for reading from or writing to a removable magnetic disk **79**, and an optical disk drive **80** for reading from or writing to a removable optical disk **81** such as a CD ROM or other optical media. The hard disk drive **77**, magnetic disk drive **78**, and optical disk drive **80** are connected to the system bus **73** by a hard disk drive interface **82**, a magnetic disk drive interface **83**, and an optical drive interface **84**, respectively. The drives and their associated computer-readable media

provide nonvolatile storage of computer readable instructions, data structures, program modules and other data for the personal computer **20**.

Although a hard disk, a removable magnetic disk **29**, and a removable optical disk **31** are described, it should be appreciated by those skilled in the art that other types of computer readable media can be used to store data. Other such media include magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROM), and the like.

A number of program modules may be stored on the hard disk, magnetic disk **79**, optical disk **81**, ROM **74**, or RAM **75**. These programs include an operating system **85**, one or more application programs **86**, other program modules **87**, and program data **88**. A user may enter commands and information into the personal computer **70** through input devices such as keyboard **90** and pointing device **92**. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit **71** through a serial port interface **96** that is coupled to the system bus **73**, but may be connected by other interfaces, such as a parallel port, gamc port, or a universal serial bus (USB). A monitor **97** or other type of display device is also connected to the system bus **73** via an interface, such as a video adapter **98**. In addition to the monitor, personal computers typically include other peripheral output devices (not shown) such as speakers and printers.

The server computer **70** is connected to the Internet **42** through a network interface or adapter **100**, a modem **102**, or other means for establishing communications over the Internet. The modem **102**, which may be internal or external, is connected to the system bus **73** via the serial port interface **96**.

FIG. **5** shows an exemplary implementation in which the store builder module **46** (or **56**) is implemented within the operating system **85** on the host server computer **70**. The server computer **70** runs a server operating system, which is preferably a Windows brand operating system from Microsoft Corporation. One preferred operating system is the Windows NT operating system. The operating system **85** includes an Internet Information Services component **110** that provides the functionality to support a wide variety of Internet services. The store builder module **46** is integrated into the Internet Information Services component **110**. The store builder module **46** includes a store builder wizard **112**, the page generator **24**, and a pipeline configurator **114**.

System Software Architecture

FIG. **6** shows the general system software architecture for the store builder module **46**, which takes input from a merchant and generates a customized Web site storefront **120**. The store builder wizard (SBW) **112** guides a merchant through a series of step by step instructions for entering data pertaining to the merchant's business. In one implementation, the SBW **112** is configured as a set of active server pages **122** which are ordered by a sequencer **124** to present a series of user interface screens that ask various questions designed to extract information from the merchant. For instance, the ASPs **122** are transformed when rendered to a series of HTML documents that require the merchant to enter data concerning the business' address, inventory, pricing, preferred method of payment, and so forth.

FIGS. **7** and **8** show examples of two SBW questionnaire pages shown on a display **150**. FIG. **7** shows the "Merchant Information" page 152, which pertains to general information about the merchant, such as business name, address, and business description. When the merchant invokes the store builder wizard **112** (either locally on their own computer, or remotely via a browser), the "Merchant Information" page 152 is rendered and the merchant enters data into the fields.

FIG. **8** shows the "Style" page 154, which is directed to storefront properties, such as background color, font color, and hyperlink color. This page is presented in the sequence after the "Merchant Information" page 152 of FIG. **7**. The merchant enters preferred color choices and continues onto the next page.

In addition to the two pages shown in FIGS. **7** and **8**, there are many other possible types of pages. Examples of possible pages include a "Welcome" page that greets the merchant; a "Locale" page for entry of currency, language, taxes, and time; a "Product" page for defining a product line; a "Shipping and Handling" page for entry of preferred shipment techniques; a "Tax" page for entry of special taxes; and a "Payment Method" page for selection of preferred methods of payment.

With reference again to FIG. **6**, the merchant's data entered to the SBW is collected and stored in a data file **126**. The data file **126** is passed to a wizard site generator **128**, which directs the data to the page generator **24** and the pipeline configurator **114**. The pipeline configurator **114** establishes business rules concerning order processing and procedures for handling purchases. The rules are generated based on the merchant data collected by the SBW **112**. The rules are stored in a data file **130** at the generated storefront **120**.

The store builder module **46** includes a library of active server templates **132** that are generic for a variety of merchant storefronts. The page generator **24** reads the active server templates **132** and uses them to generate a set of one or more active server pages **134** and an active server application **136** while using the merchant data in the data file **126** as input to the templates. The generation of the ASPs **134** from the templates **132** and merchant data **126** is achieved in the same manner described above with reference to FIGS. **1** and **2**.

The page generator **24** can also generate database-compatible files that might be used to store products and services available from the merchant. In this example, the page generator **24** creates a SQL file **138** that is compatible with SQL server database software from Microsoft Corporation. The SQL file **138** is loaded by loader **140** into a database **142** to serve as a product/service database supporting the merchant's storefront.

The store builder module also enables the merchant to make changes to their storefront without having to create an entirely new storefront. For instance, suppose the merchant wants to change the color of the background from blue to green. The merchant invokes the store builder wizard and sequences to the "Style" page **154** in FIG. **8**. The merchant then changes the background field from "blue" to "green". This change is stored in the data file **126** and the page generator **24** generates a new ASP that reflects the background color change. Accordingly, the merchant can make occasional changes to its storefront, without needing the programming skills to directly modify the underlying pages.

Although the invention has been described in language specific to structural features and/or methodological steps, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or steps described. Rather, the specific features and steps are disclosed as preferred forms of implementing the claimed invention.

What is claimed is:

1. A computer-implemented system comprising:

an active server template written in a combination of a hypertext language and a scripting language, the active server template having two levels of scripting language that are denoted by first and second delimiters; and

a page generator to execute the active server template by executing one level of the scripting language denoted by the first delimiter to produce at least one active server page, the active server page containing the hypertext language and the other level of scripting language denoted by the second delimiter.

2. A computer-implemented system as recited in claim 1, further comprising an operating system embodied in a computer-readable medium, wherein the page generator is incorporated into the operating system.

3. A computer-readable program language embodied on a computer-readable medium comprising:

hypertext code for specifying hypertext terms; and

scripting code for providing scripting functionality to form a hypertext page using the hypertext terms, the scripting code having first and second levels of instructions denoted by first and second delimiters whereupon execution of the scripting code's first level of instructions denoted by the first delimiter results in an executable structure containing the hypertext code and the scripting code's second level of instructions and subsequent execution of the scripting code's second level of instructions results in a renderable structure containing hypertext code.

4. A computer-readable program language as recited in claim 3, wherein the hypertext code comprises hypertext markup language (HTML).

5. A computer-readable template embodied on a computer-readable medium comprising:

hypertext terms that can be rendered by a browser; a first level of scripting code that upon execution yields a hypertext page having the hypertext terms, the first level of scripting code being denoted by a first delimiter; and

a second level of scripting code that upon execution yields an active server page containing the hypertext terms and the first level of scripting code, the second level of scripting code being denoted by a second delimiter different from the first delimiter to enable execution of the second level of scripting code independent of the first level of scripting code.

6. A computer operating system embodied on a computer-readable medium, the operating system comprising a page generator to convert an active server template to an active server page that can be rendered by an Internet browser, the active server template being written in a combination of a hypertext language and a scripting language, the page generator converting the active server template to a scripting program and executing the scripting program to produce the active server page.

7. A computer operating system as recited in claim 6, whereby the scripting language of the active server template is configured with first and second levels of scripting language that are denoted by first and second delimiters, the page generator executing the first level of the scripting language denoted by the first delimiter to produce the active server page.

8. A computer operating system as recited in claim 6, whereby the scripting language of the active server template is configured with first and second levels of scripting language that are denoted by first and second delimiters, the page generator converting the first level of the scripting language denoted by the first delimiter to print statements that form the scripting program.

9. A method for creating a Web site comprising the following steps:

collecting data pertaining to the Web site;

reading one or more Web page templates that are generic for a variety of Web sites; and

generating a set of one or more Web pages that form the Web site based on the site data and the set of generic Web page templates.

10. A method as recited in claim 9, wherein the Web page templates comprise active server templates, each active server template containing a hypertext language and a scripting language, the scripting language having first and second levels of instructions denoted by first and second delimiters, the step of generating comprises the following steps:

converting the active server templates to a scripting program having executable lines of code derived from the first level of instructions denoted by the first delimiters; and

executing the scripting program to produce the Web pages, the Web pages containing the hypertext language and the second level of instructions in the scripting language denoted by the second delimiters.

11. A method as recited in claim 9, wherein the data collecting step comprises the step of presenting step by step instructions to a user for entering the data.

12. A method as recited in claim 9, wherein the data collecting step comprises the step of presenting a series of user interface screens that enable a user to respond to various questions, the responses being collected as the data.

13. A method as recited in claim 9, wherein the steps of collecting, reading, and generating are performed at a first computing location, and further comprising the step of submitting the data for collection from an online computing location that is connected to, but remote from, the first location.

14. A computer-readable medium comprising computer-executable instructions for performing the steps in the method as recited in claim 9.

15. A method for converting an active server template to an active server page, comprising the following steps:

reading a file containing the active server template, the active server template containing a hypertext language and a scripting language, the scripting language having first and second levels of instructions denoted by first and second delimiters;

converting the active server template file to a scripting program having executable lines of code derived from the first level of instructions denoted by the first delimiters; and

executing the scripting program to produce an active server page containing the hypertext language and the second level of instructions in the scripting language denoted by the second delimiters.

16. A method as recited in claim 15, further comprising the step of storing the active server page in a data file.

17. A computer-readable medium comprising computer-executable instructions for performing the steps in the method as recited in claim 15.

* * * * *

Exhibit E

US006122558A

United States Patent [19]

Barnes et al.

[11] **Patent Number: 6,122,558**

[45] **Date of Patent: Sep. 19, 2000**

[54] **AGGREGATION OF SYSTEM SETTINGS INTO OBJECTS**

[75] Inventors: **David A. Barnes**; **Joyce A. Grauman**; **Renee Marceau**; **Virginia E. S. Howlett**, all of Seattle; **Kevin Schofield**, Bellevue; **Mark A. Malamud**, Seattle; **Issac J. Heizer**, Woodinville; **Daniel F. E. Plastina**, Redmond; **Chris E. Tobey**, Seattle; **Rosanne M. Lehmann**, Seattle; **William T. Flora**, Seattle; **Eric L. Van Doren**, Seattle, all of Wash.

[73] Assignee: **Microsoft Corporation**, Redmond, Wash.

[21] Appl. No.: **08/935,158**

[22] Filed: **Sep. 22, 1997**

Related U.S. Application Data

[63] Continuation of application No. 08/366,058, Dec. 29, 1994, abandoned.

[51] **Int. Cl.**[7] **G05B 11/01**

[52] **U.S. Cl.** **700/83**; 700/17

[58] **Field of Search** 700/83, 84, 17; 345/353, 352, 347, 333

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,896,291	1/1990	Gest et al.	345/353
5,455,378	10/1995	Paulson et al.	84/610
5,542,039	7/1996	Brinson et al.	395/161
5,682,490	10/1997	Sumino et al.	345/352
5,821,932	10/1998	Pittore	345/347

OTHER PUBLICATIONS

Microsoft® Windows™ and MS–DOS® 6, User's Guide, Microsoft Corporation; 1993; Chapter 12, "Customizing Windows," pp. 122–139.

Microsoft® Windows NT™ Advanced Server, Version 3.1, System Guide, Microsoft Corporation; 1993; "Overview," pp. 112–113 and 132–133.

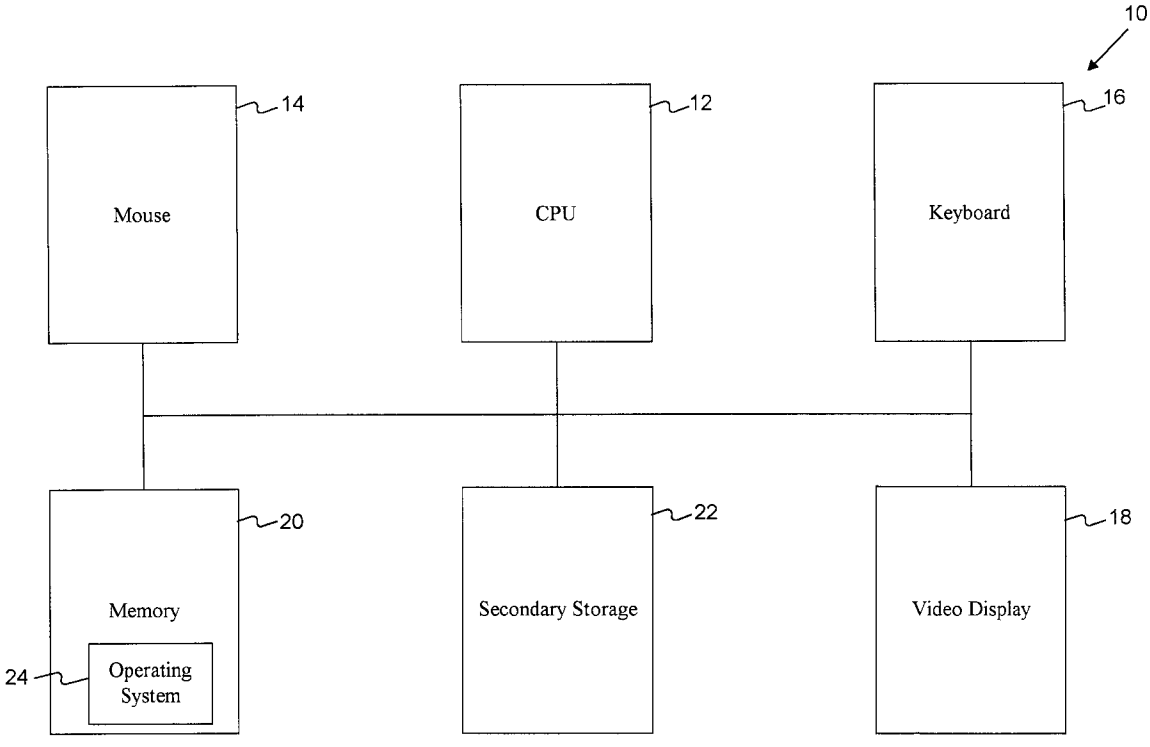
Primary Examiner—Paul P. Gordon

Attorney, Agent, or Firm—Banner & Witcoff, Ltd.

[57] **ABSTRACT**

A control panel provides controllers for setting the values of system settings. Each controller controls a subset of related system settings. Scheme objects are provided for encapsulating values of system settings for a controller. The current values of system settings controlled by a controller may be updated by applying the values held within a scheme. Grand schemes are provided for aggregating the system settings for multiple controllers. Thus, the system settings controlled by multiple controllers may be updated in a single transaction by applying a grand scheme to the control panel. Easily practiced approaches to applying schemes and grand schemes to the control panel are provided. Moreover, methods for easily creating and editing the contents of schemes and grand schemes are provided.

42 Claims, 18 Drawing Sheets



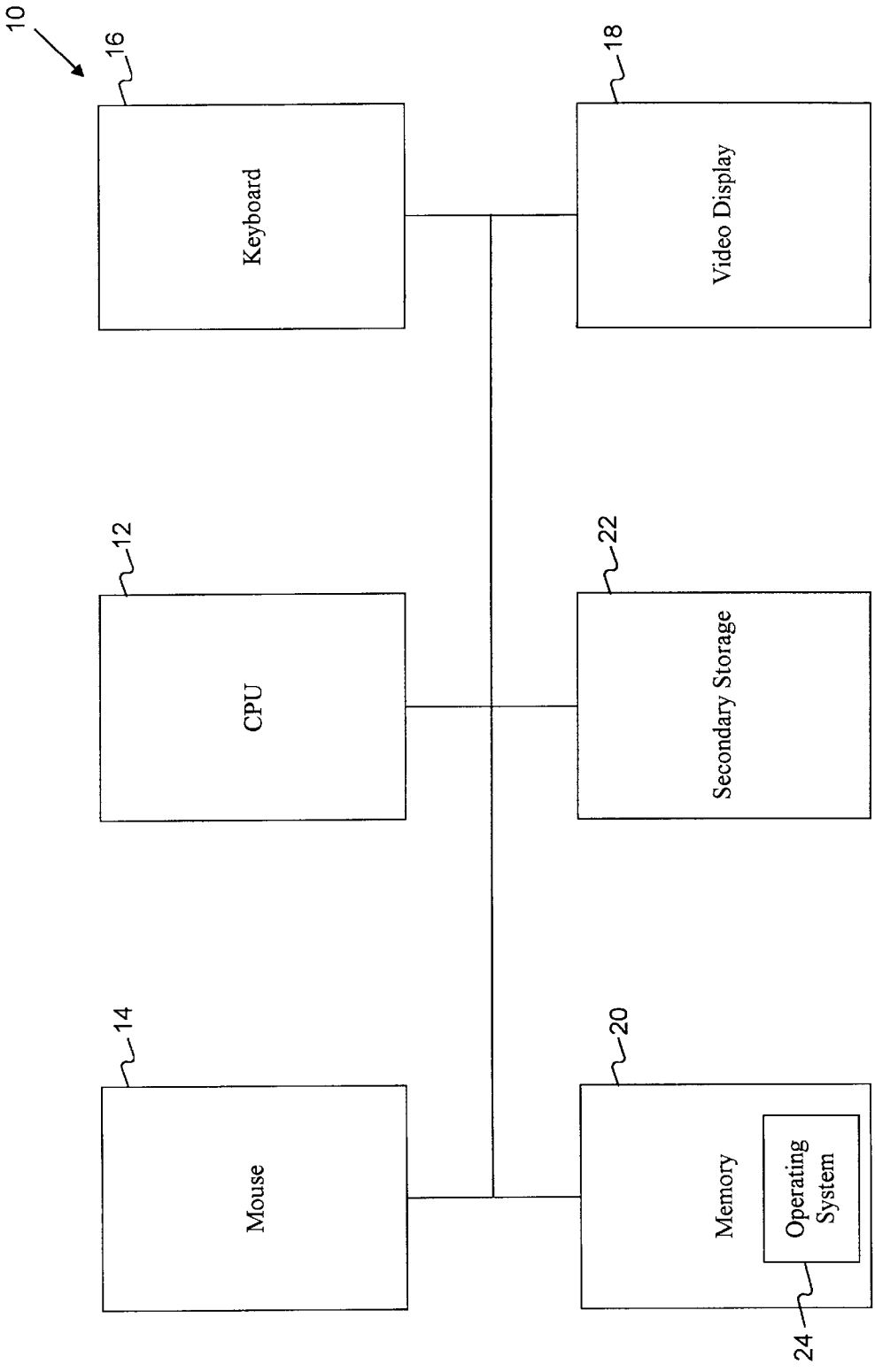

10
16
Keyboard
18
Video Display
12
CPU
22
Secondary Storage
14
Mouse
20
Memory
Operating System
24

FIG. 1

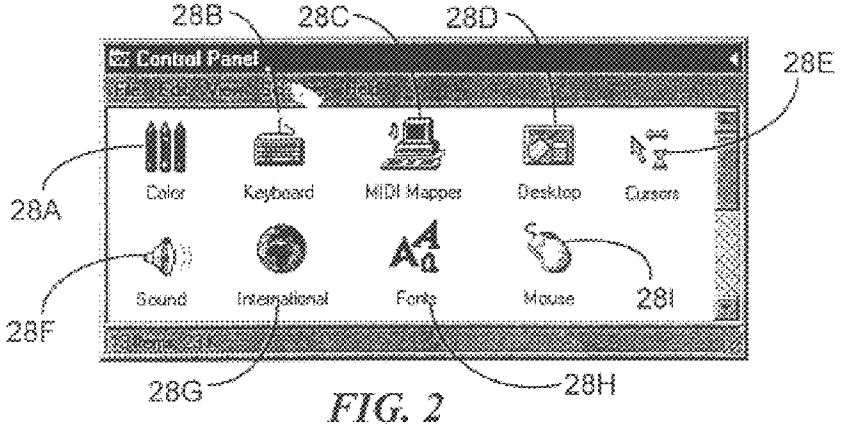


FIG. 2

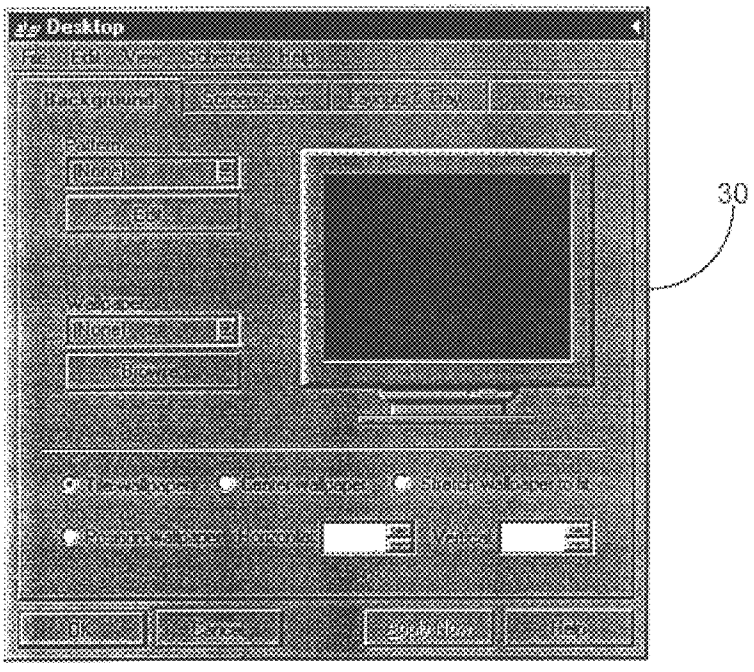


FIG. 3

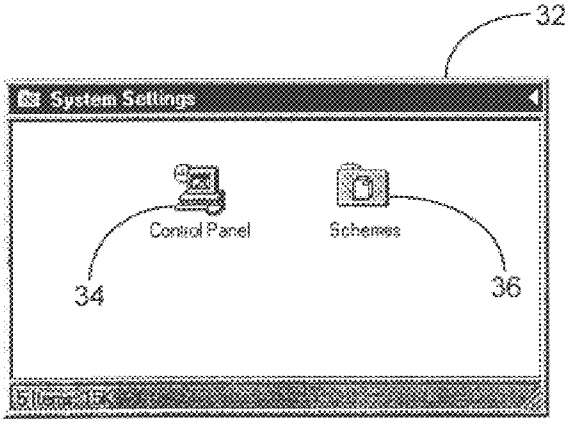


FIG. 4

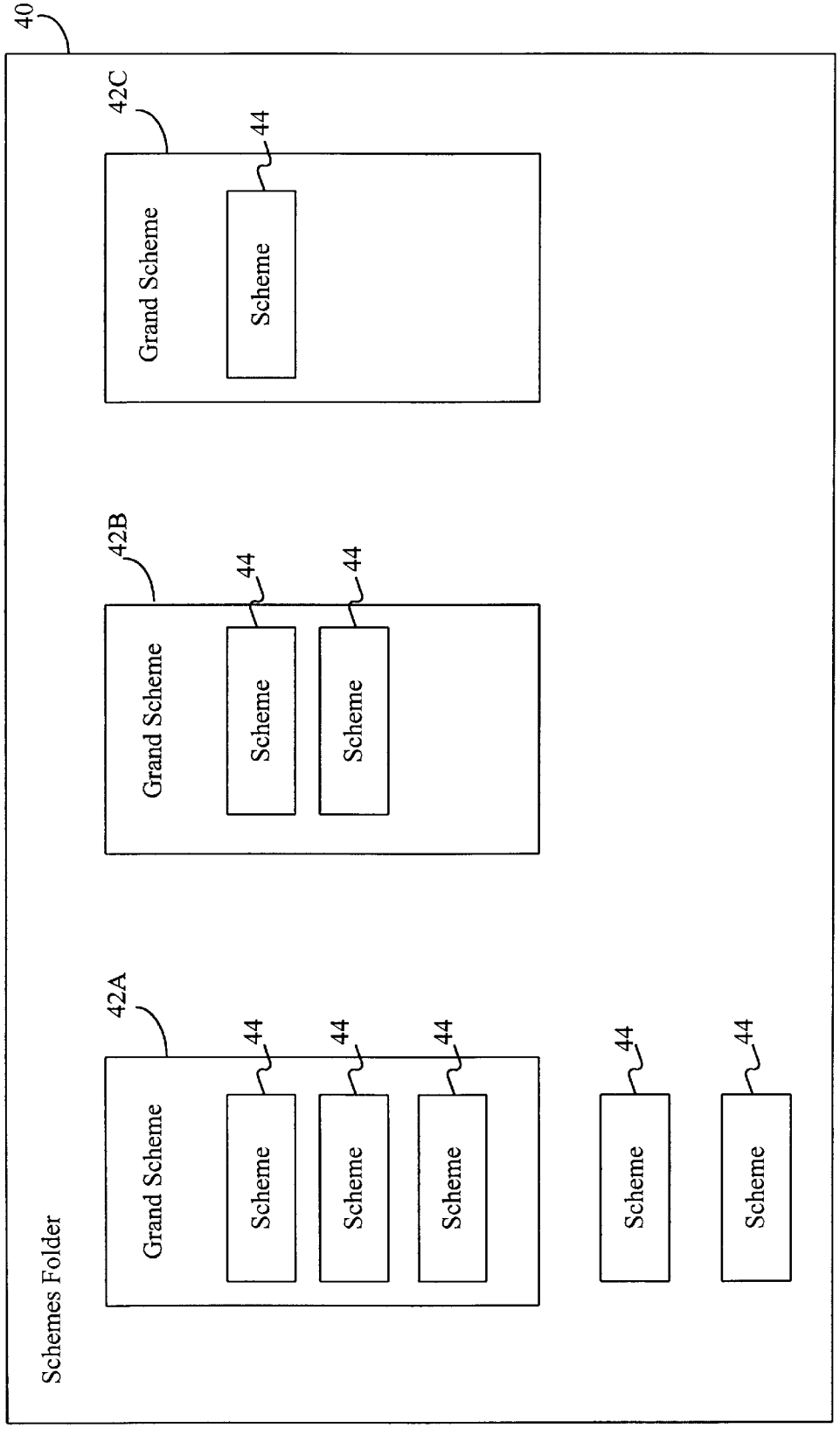

40
Schemes Folder
42A
Grand Scheme
44
Scheme
42B
Grand Scheme
42C
Grand Scheme

FIG. 5

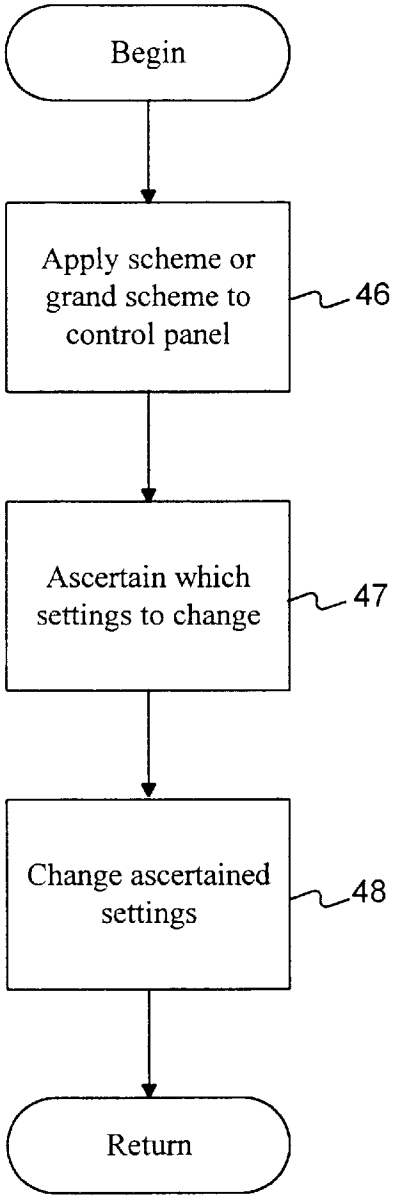

Begin
Apply scheme or grand scheme to control panel
46
Ascertain which settings to change
47
Change ascertained settings
48
Return

FIG. 6

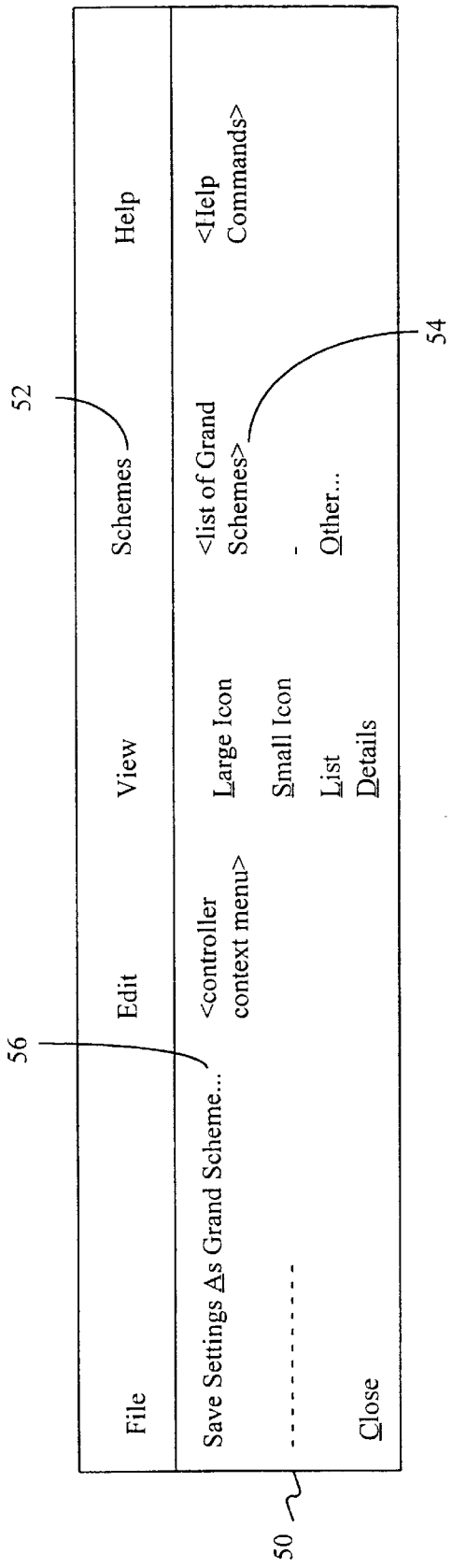

File
Edit
View
Schemes
Help
Save Settings As Grand Scheme...
<controller context menu>
Large Icon
<list of Grand Schemes>
<Help Commands>
Small Icon
-
List
Other...
Details
Close
50
52
54
56

FIG. 7A

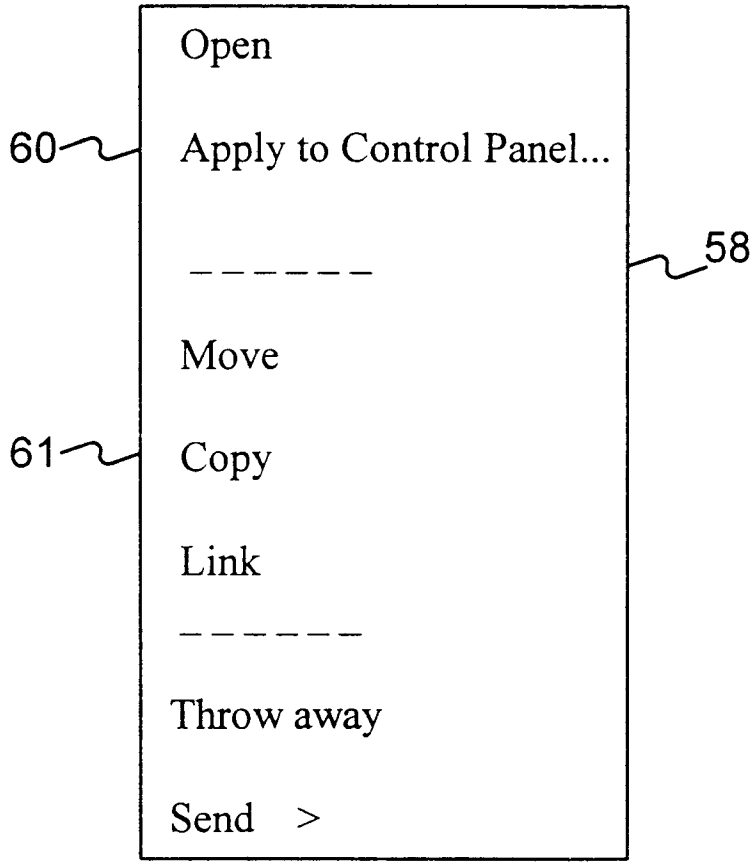

Open
Apply to Control Panel...
60
58
Move
Copy
61
Link
Throw away
Send >

FIG. 7B

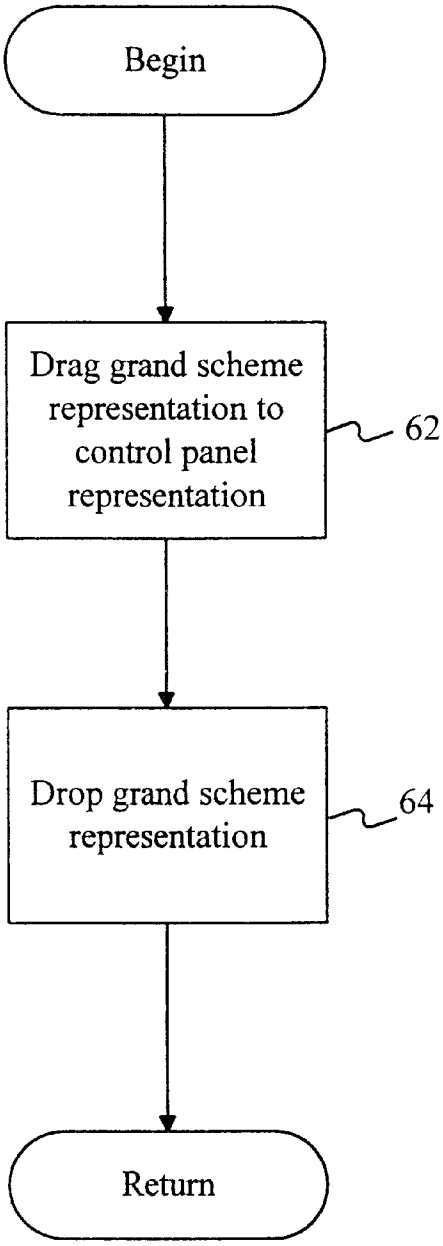
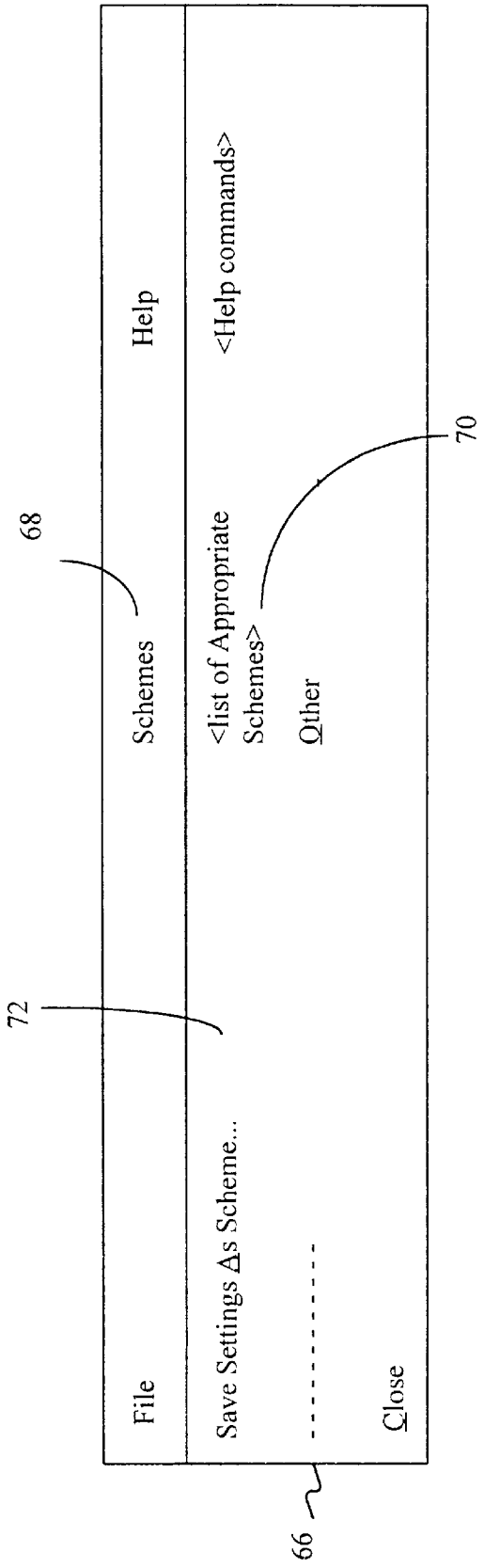

Begin
Drag grand scheme representation to control panel representation
62
Drop grand scheme representation
64
Return

FIG. 7C


File
Save Settings As Scheme...

Close
Schemes
Other
Help
<Help commands>
<list of Appropriate Schemes>
66
68
70
72

FIG. 8A

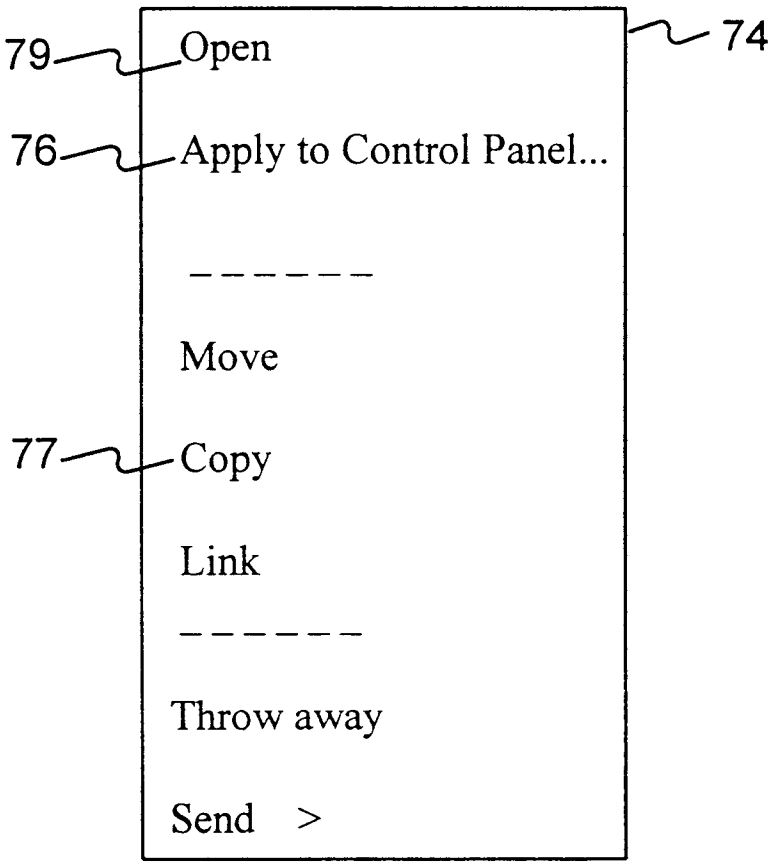

79
Open
74
76
Apply to Control Panel...
Move
77
Copy
Link
Throw away
Send >

FIG. 8B

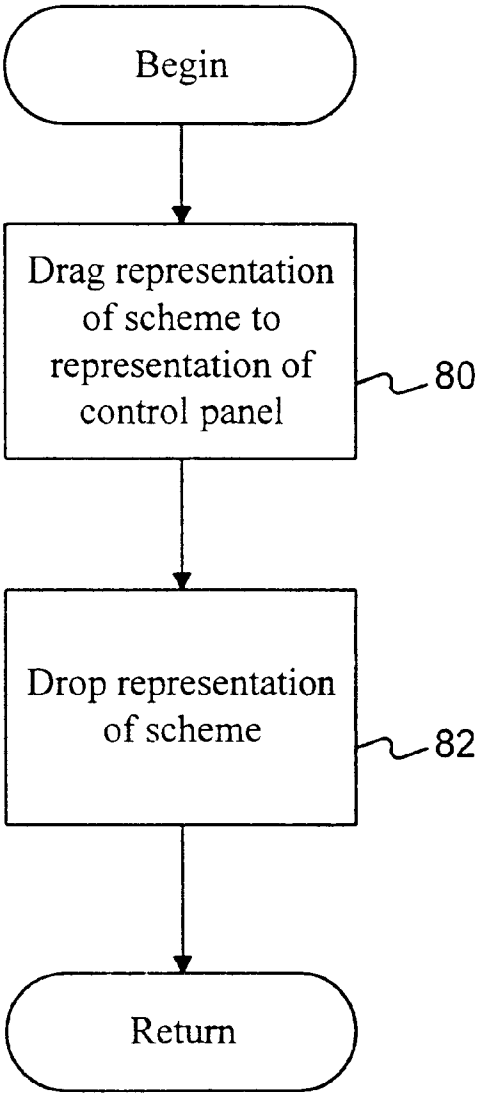

Begin
Drag representation of scheme to representation of control panel
80
Drop representation of scheme
82
Return

FIG. 8C

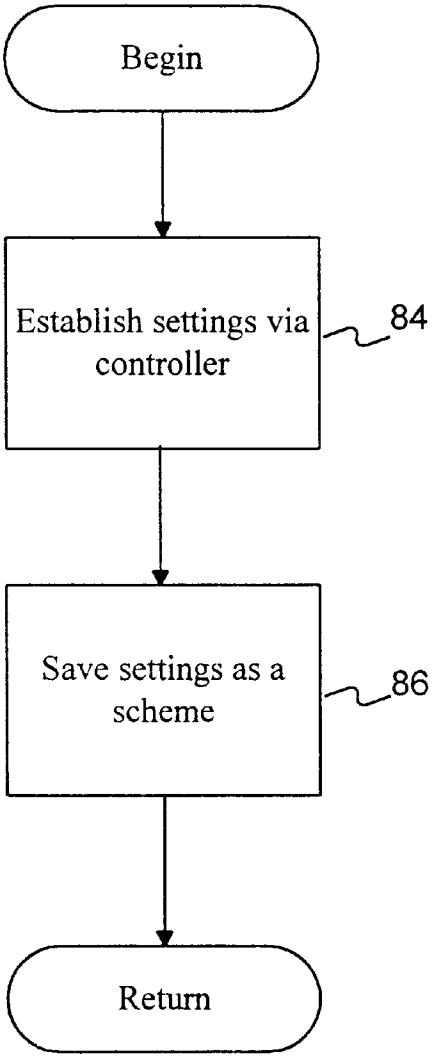

Begin
Establish settings via controller
84
Save settings as a scheme
86
Return

FIG. 9A

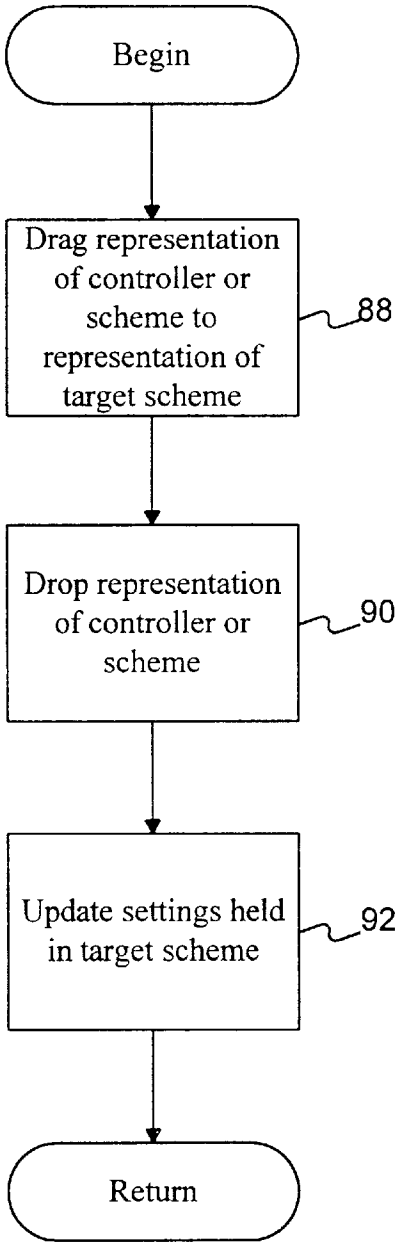

Begin
Drag representation of controller or scheme to representation of target scheme
88
Drop representation of controller or scheme
90
Update settings held in target scheme
92
Return

FIG. 9B

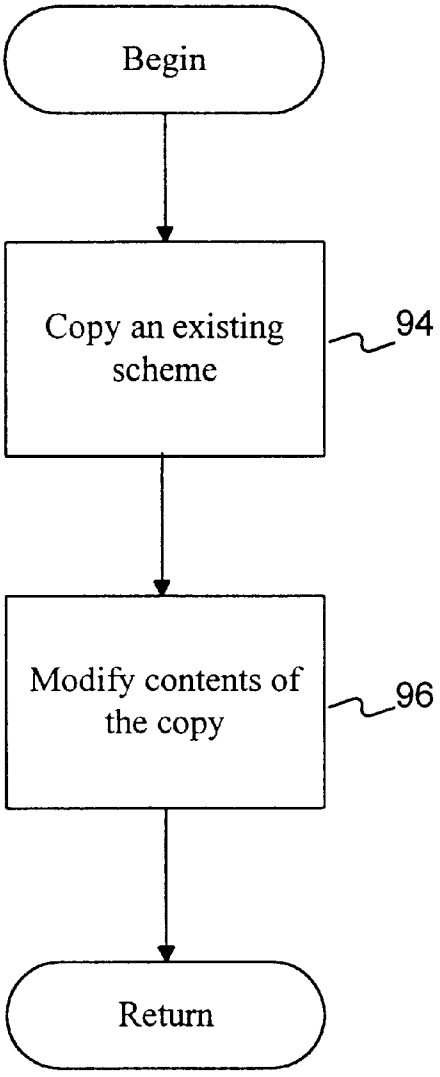

Begin
Copy an existing scheme
94
Modify contents of the copy
96
Return

FIG. 9C

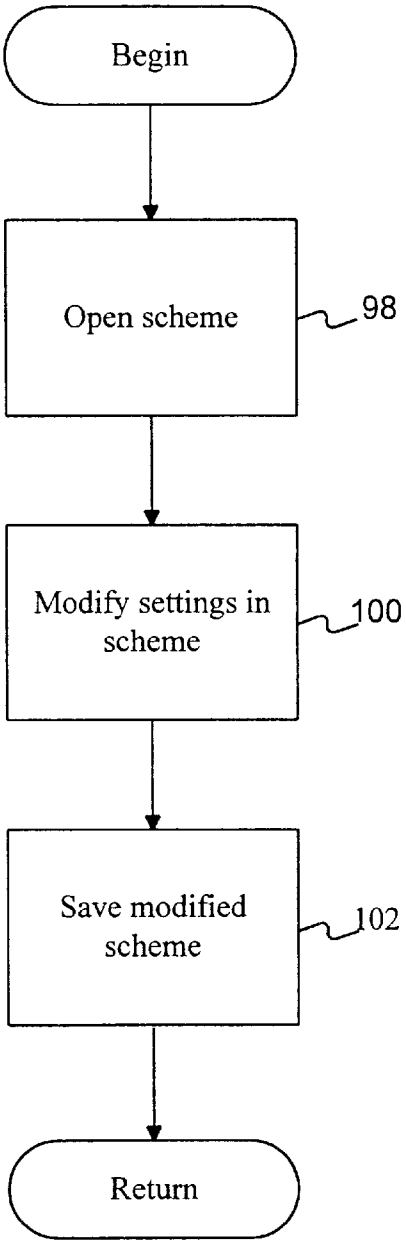

Begin
Open scheme
98
Modify settings in scheme
100
Save modified scheme
102
Return

FIG. 10

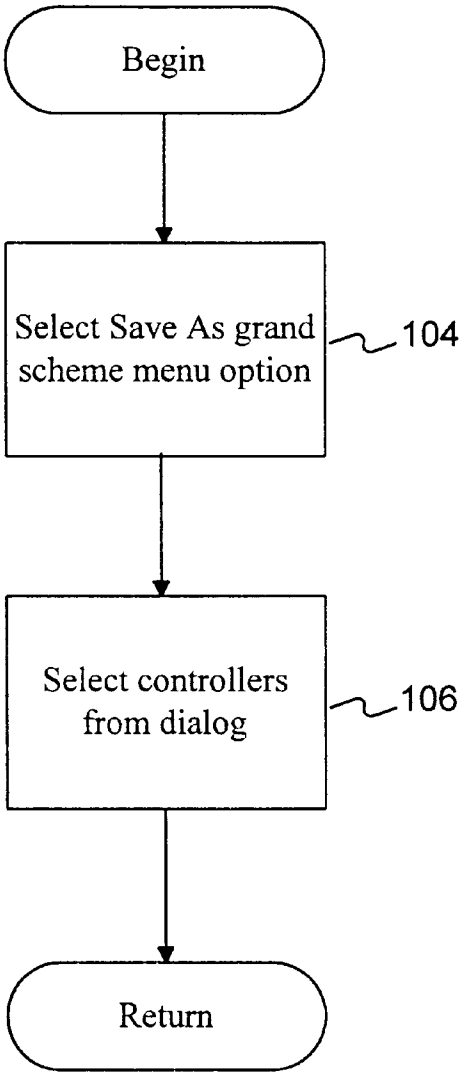

Begin
Select Save As grand scheme menu option
104
Select controllers from dialog
106
Return

FIG. 11A

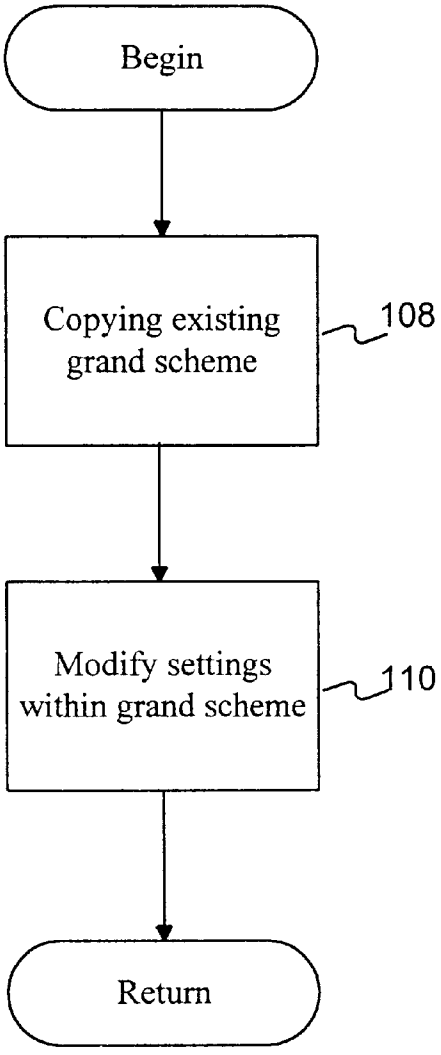

Begin
Copying existing grand scheme
108
Modify settings within grand scheme
110
Return

FIG. 11B

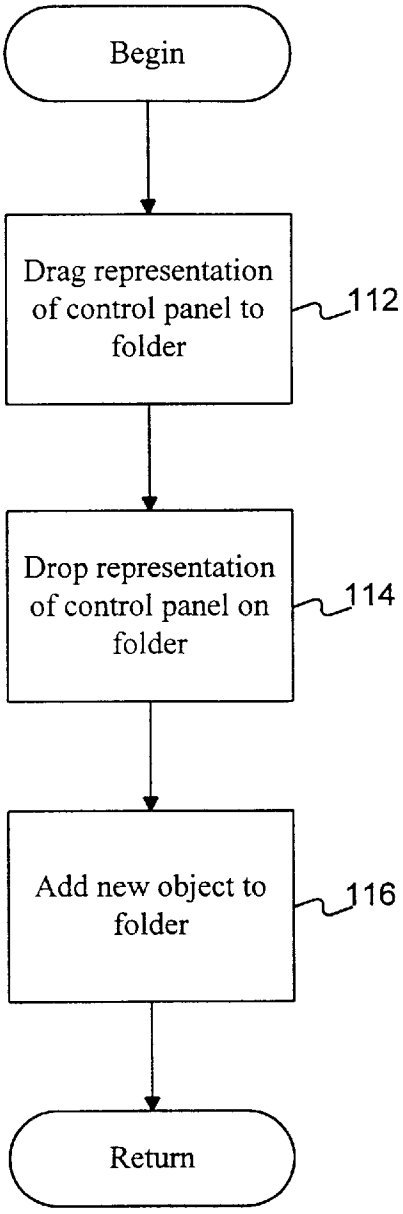

Begin
Drag representation of control panel to folder
112
Drop representation of control panel on folder
114
Add new object to folder
116
Return

FIG. 11C

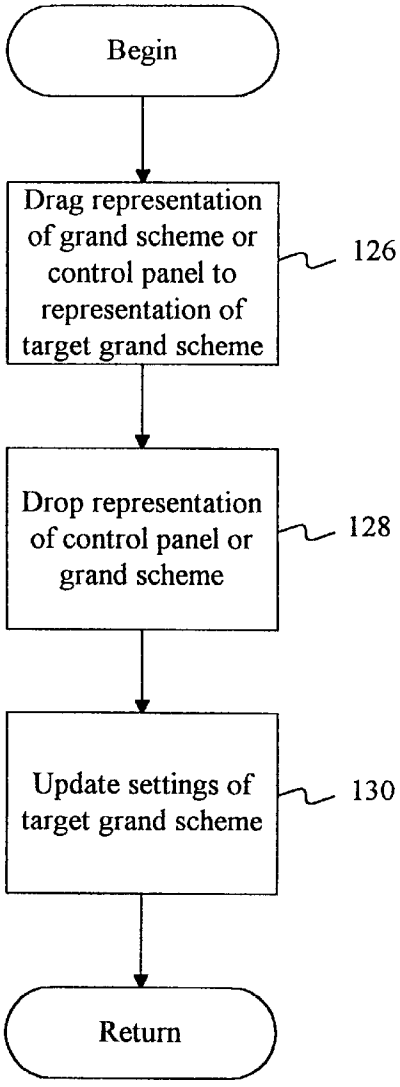

Begin
Drag representation of grand scheme or control panel to representation of target grand scheme
126
Drop representation of control panel or grand scheme
128
Update settings of target grand scheme
130
Return

FIG. 12

AGGREGATION OF SYSTEM SETTINGS INTO OBJECTS

CROSS-REFERENCE TO RELATED APPLICATION

This application is a continuation of U.S. patent application Ser. No. 08/366,058, filed Dec. 29, 1994, now abandoned.

TECHNICAL FIELD

This invention relates generally to data processing systems and, more particularly, to system settings within data processing systems that control an operating environment.

BACKGROUND OF THE INVENTION

The Microsoft WINDOWS, version 3.1, operating system, sold by Microsoft Corporation of Redmond, Wash., provides a control panel that allows a user to adjust various system settings, such as the color settings for the graphical user interface. The control panel includes a number of controllers that adjust groups of settings. For example, a separate color controller is provided to adjust the color settings that are used by the operating system. Each controller generates a dialog box when activated that allows a user to select the system settings that are controlled by the controller. The operating system provides default settings for each of the system settings that are controlled by the controllers. However, in order to change the system settings, the user must activate each controller in turn to adjust system settings which the user wishes to change.

SUMMARY OF THE INVENTION

In accordance with the first aspect of the present invention, a method is practiced in a computer system having a video display and a storage device. The computer system runs an operating system that provides a desktop environment to a user and a file system. The desktop environment has associated systems settings that affect the desktop environment. In this method, a first set of values for at least a portion of the system settings are stored in the storage device so that the first set of values is visible in the file system. A second set of values, for the same portion of the system settings for which values are stored in the first set of values, is also stored in the storage device such that the second set of values is visible in the file system. In response to a choice by the user between the first set of values and the second set of values, the system settings are updated to have the values specified by the chosen set of values.

In accordance with another aspect of the present invention, a control panel is provided as part of an operating system that is run on a computer system. The control panel is used to control values assigned to system settings that control an operating environment provided to a user. Sets of values for the system settings are stored in the storage device. Each set of values includes values for at least a portion of the system settings. The user selects ones of the sets of values via a provided interface, and the current system settings are changed to have the values of the selected set of values.

In accordance with a further aspect of the present invention, a control panel is provided for controlling current values of the system settings. The control panel includes controllers that are each responsible for controlling current values of a group of related ones of the system settings. The sets of values are visible in a file system of the operating system. The user is provided with at least two sets of values for an identified one of the groups of related system settings. The user selects one of the two sets of values, and in response to the user selection, the current values of the identified group of related system settings are changed to the values of the set selected by the user.

In accordance with an additional aspect of the present invention, a grand scheme container object is stored in a storage device. Scheme objects are stored within the grand scheme container object. Each scheme object holds a set of values for a subset of the system settings. The values held in the scheme objects that are contained within the grand scheme container object are applied to the current system settings so that the current system settings assume the values held in the scheme objects.

In accordance with another object of the present invention, a method is practiced in a computer system having an input device and a video display. The computer system runs an operating system that provides an operating environment to a user as specified by system settings. A first object holds values for system settings and has a representation that is displayed on the video display. A second object also holds system settings and also has a representation on the video display. The representation of the first object is dragged to lie over at least a portion of the representation of the second object in response to the user using the input device. The representation of the first object is dropped on the representation of the second object. In response to the dropping of the representation of the first object in the representation of the second object, the values for system settings are changed in the second object to the values contained within the first object.

In accordance with a further aspect of the present invention, a data processing system includes a processor for running an operating system. The operating system provides a file system and a desktop environment to the user. The desktop environment has an associated set of system settings that affect the environment. The data processing system also includes storage. The storage holds a copy of the operating system and a first and second set of values. The sets of values for a portion of the system settings are visible within the file system. The data processing system provides a vehicle for updating the system settings in response to the user choice of one of the sets of values such that the system settings assume the values of the set of values chosen by the user.

In accordance with a final aspect of the present invention, a system for providing a desktop environment to a user is provided. The desktop environment has an associated set of system settings that affect the desktop environment. The system includes a display component for displaying an interface to a user as part of the desktop environment. The display component displays the interface according to system settings. The system also includes a first container holding a first set of system setting values and a second container holding a second set of system setting values. The system additionally includes a selection component that receives selection information, and in response to this selection information, selects between the first container and the second container. A change component is provided as part of the system that is responsible to the selection made by the selection component to change the system settings to those held in the selected container.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a computer system that is suitable for practicing a preferred embodiment of the present invention.

FIG. **2** is a diagram illustrating an example control panel window in accordance with the preferred embodiment of the present invention.

FIG. **3** is an example of a dialog box for a controller for the control panel of FIG. **2**. FIG. **4** is a diagram illustrating an example of a default systems setting window provided by the preferred embodiment of the present invention.

FIG. **5** is a diagram illustrating the hierarchy among a schemes folder, grand schemes and schemes in accordance with the preferred embodiment of the present invention.

FIG. **6** is a flow chart illustrating how schemes and grand schemes are used to change system settings.

FIG. **7**A illustrates a menu for a control panel in accordance with the preferred embodiment of the present invention.

FIG. **7**B illustrates an example of a context menu for a grand scheme.

FIG. **7**C illustrates steps performed to drag and drop a grand scheme on a control panel in accordance with the preferred embodiment of the present invention.

FIG. **8**A is a diagram illustrating an example of a menu for a controller in accordance with the preferred embodiment of the present invention.

FIG. **8**B is a diagram illustrating an example of a context menu for a scheme.

FIG. **8**C is a flow chart illustrating the steps performed to drag-and-drop a scheme onto the control panel in the preferred embodiment of the present invention.

FIG. **9**A is a flow chart illustrating the steps of a first approach for creating a scheme in accordance with the preferred embodiment of the present invention.

FIG. **9**B is a flow chart illustrating the steps of a second approach for creating a scheme in accordance with the preferred embodiment of the present invention.

FIG. **9**C is a flow chart illustrating the steps of a third approach for creating a scheme in accordance with the preferred embodiment of the present invention.

FIG. **10** is a flow chart illustrating the steps performed to edit a scheme in accordance with the preferred embodiment of the present invention.

FIG. **11**A is a flow chart illustrating the steps of a first approach for creating a grand scheme in accordance with the preferred embodiment of the present invention.

FIG. **11**B is a flow chart illustrating the steps of a second approach for creating a grand scheme in accordance with the preferred embodiment of the present invention.

FIG. **11**C is a flow chart illustrating the steps of an approach for creating an object holding system settings of a control panel in accordance with the preferred embodiment of the present invention.

FIG. **12** is a flow chart illustrating the steps of a second approach for modifying a grand scheme in the preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The preferred embodiment of the present invention provides a user with a quick and easy way to change groups of system settings in one transaction. Specifically, the preferred embodiment of the present invention supports the use of schemes and grand schemes on a per user per desktop basis. Schemes are entities that hold collections of system settings for a particular controller of a control panel (i.e., a control panel applet). Grand schemes are collections of settings for a set of one or more controllers. Through the use of schemes or grand schemes, a user can update a group of settings associated with a controller or group of controllers. The user merely needs to specify the scheme or grand scheme that is to be used to establish the settings and then request that the scheme or grand scheme be applied. The user may store a number of different schemes and grand schemes within the system. The controllers, schemes, and grand schemes are all implemented as objects that may be dragged and dropped to establish and change system settings.

FIG. **1** is a block diagram of a computer system **10** that is suitable for practicing the preferred embodiment of the present invention. The computer system includes a central processing unit (CPU) **12** that controls the activities of the computer system. The CPU **12** may be a microprocessor or other type of commercially available CPU. The computer system **10** also includes input devices, such as mouse **14** and keyboard **16**. A video display **18** is provided in the computer system **10** to display video output to the user. A memory **20** and a secondary storage device **22** provide storage facilities within the computer system **10**. The secondary storage device **22** may be a hard disk drive or other suitable secondary storage device. The memory **20** holds a copy of an operating system **24**. In the preferred embodiment of the present invention, the operating system **24** is used to store and manage the control panel, the schemes, and the grand schemes.

In order to understand the use of schemes and grand schemes within the preferred embodiment of the present invention, it is helpful to first review the role the control panel serves within the computer system **10**. As was discussed in the Background of the Invention, the control panel is provided by the operating system **24** to enable the user to adjust system settings. FIG. **2** shows an open control panel window that includes a separate icon **28**A–**28**I for each controller. The controllers in FIG. **2** include a color controller **28**A, a keyboard controller **28**B, a MIDI mapper controller **28**C, a desktop controller **28**D, a cursors controller **28**E, a sound controller **28**F, an international controller **28**G, a fonts controller **28**H and a mouse controller **28**I.

The controllers are sections of code that control the system settings for groups of related settings. For example, the desktop controller controls the appearance of the virtual desktop to the user. When the user positions a mouse cursor through the use of mouse **14** to point at one of the icons **28**A–**28**I shown in the control panel window **26** and then double clicks a button of the mouse, a dialog box for adjusting the system settings of the associated controller is displayed. FIG. **3** shows an example of the dialog box **30** that is displayed when a user double clicks on the desktop controller icon **28**. The dialog box **30** includes a number of list boxes, option boxes, and other user interface controls that allow a user to select the settings that are controlled by the controller.

The preferred embodiment of the present invention is employed using the Microsoft Object Linking and Embedding (OLE) 2.0 protocol of Microsoft Corporation. In order to understand the implementation details of the preferred embodiment of the present invention, it is useful to first review some of the concepts employed in Microsoft OLE 2.0 that are relevant to the present invention.

Microsoft OLE 2.0 is a protocol that follows a specific object model. An "object" in Microsoft OLE 2.0 is a data abstraction that encapsulates related behavior and attributes. Typically, an object includes a number of functions and data structures. Nevertheless, in certain instances an object may

include only functions. A "container object" is an object that contains other objects within it.

An "interface" in Microsoft OLE 2.0 is a group of semantically related functions that are organized into a named unit (the name being the identifier of the interface). Interfaces have no instantiation per se (i.e., the definition for an interface does not include code for implementing the functions that are identified within an interface); rather, interface definitions merely specify a set of signatures for identified functions. "Instantiation" refers to the process of creating in memory structures that represent an object so that operations can be evoked on the object. When an object "supports" an interface, the object provides code for the functions specified by the interface. Thus, the objects that support the interfaces are responsible for providing the code for implementing the functions of the interfaces. The code provided by an object must comply with the signatures of the interface definition. The run-time manifestation of an interface instance is a data structure that provides access to the functions defined for the interface. Interface instances are referenced by clients of a server object using interface pointers. Hence, when a client object is provided with an interface pointer, the client object is able to invoke the functions of the instance of interface that is provided by the server object.

Schemes, controllers, and grand schemes are all implemented as objects within the preferred embodiment of the present invention. These objects support certain interfaces that will be described in more detail below. Microsoft OLE 2.0 supports the notion of "object classes." An object class groups together objects having similar properties and common behavior.

A scheme in the preferred embodiment of the present invention is an object that is a document for containing the settings of a particular controller. For example, a scheme for the desktop controller may include settings that specify the desktop pattern, screen saver parameters, wallpaper patterns, sizing grid parameters, spacing between icons, and cursor click rate. Each controller is an object that provides code for implementing the dialog box and displaying the icon associated with the controller.

A grand scheme is a special type of container object that contains one or more scheme objects. The grand scheme is well-suited for encapsulating system settings that produce a desktop of a given motif. For instance, a grand scheme might encapsulate system settings that produce a cowboy motif. Thus, the desktop settings in the grand scheme might include a wallpaper with cars on it and a rustic pattern. Moreover, the sound settings in the grand scheme may include cowboy cries and the sound of hoofs. The preferred embodiment of the present invention implements a grand scheme as an object of the grand scheme class. Similarly, controllers and schemes are implemented as objects of controller and scheme object classes, respectively. All of these objects have associated icons and windows.

In the preferred embodiment of the present invention, the control panel and the schemes are held within a system settings folder. When the system settings folder is opened, a system settings window **32**, as shown in FIG. **4**, is displayed. An icon **34** for the control panel and an icon **36** for the schemes are displayed within the system settings window. When a user double clicks on the icon **34** for the control panel, the user sees the control panel window **26** shown in FIG. **2**. When the user double clicks on the schemes icon **36**, the schemes and grand schemes stored within the schemes folder become visible.

A schemes folder is associated with each virtual desktop. Thus, different users that share a single machine may have separate sets of schemes and grand schemes or share such schemes or grand schemes. Moreover, schemes and grand schemes are stored on a per desktop basis such that a user having multiple virtual desktops may have a separate set of schemes and grand schemes for each virtual desktop. The schemes and grand schemes that are objects visible in the file system provided by the operating system.

FIG. **5** is a block diagram that illustrates the contents of an example schemes folder **40**. This schemes folder holds grand schemes **42**A, **42**B, and **42**C, and each of the grand schemes contains one or more schemes **44**. The schemes folder **40** also holds additional schemes **44** that are not contained within any grand schemes. Those skilled in the art will appreciate that the contents of the schemes folder **40** shown within FIG. **5** are merely illustrative. The schemes folder **40** may hold different contents, including objects that are neither schemes nor grand schemes.

Once the schemes or grand schemes are created, they may be used to adjust the settings within the control panel. FIG. **6** shows a flow chart of the steps performed to adjust the settings within the control panel using schemes or grand schemes. Initially, the scheme or grand scheme is applied to the control panel (step **46**). As will be described in more detail below, there are a number of different vehicles for applying the schemes or grand schemes to the control panel. A determination of which settings to change is made (step **47**). It is worth recalling from above that not all settings in a grand scheme are necessarily applied and that not all control panel settings are subject to being changed. Once the settings to be changed have been ascertained, the settings of the control panel are changed to reflect those settings marked as applicable that are encapsulated within the scheme or grand scheme being applied (step **48**). There is flexibility in that certain system settings within a scheme or grand scheme may be designated as not applicable. Such system settings are not applied when the scheme or grand scheme is applied.

The preferred embodiment of the present invention provides multiple mechanisms for applying grand scheme to the control panel (see step **46** in FIG. **6**). Those skilled in the art will appreciate that these three approaches are not intended to be exhaustive or limiting of the present invention. With reference to FIG. **7**A, a first approach to applying a grand scheme to the control panel is via menu bar **50** that is provided by the control panel. The menu bar **50** provided within the control panel includes a "Schemes" option **52** that when activated displays a list of grand schemes **54**. The user may choose any one of the grand schemes displayed within the list **54** to apply the settings of the selected grand scheme to the control panel.

A second option provided by the preferred embodiment of the present invention to apply a grand scheme to the control panel is provided in a context menu **58** (FIG. **7**B) of a grand scheme. The context menu **58** is displayed by double clicking on an icon for the grand scheme using mouse **14**. One of the options provided within the context menu **58** is option **60** to apply the grand scheme to the control panel.

The third option for applying a grand scheme to the control panel in the preferred embodiment of the present invention is to perform a drag-and-drop operation. FIG. **7**C shows the steps that are performed in such an operation. In particular, a representation of the grand scheme (such as an open grand scheme window or an icon for a grand scheme) is dragged using the mouse **14** to be over a control panel

representation (such as control panel icon or an open control panel window) (step **62** in FIG. **7**C). The grand scheme representation is then dropped (step **64**) so as to apply the grand scheme to the control panel.

Hence, it can be seen that the preferred embodiment provides a number of easily implemented ways to update control panel settings with grand schemes. Multiple approaches are provided to suit the user's preference.

Three approaches are provided by the preferred embodiment of the present invention to apply schemes, as opposed to grand schemes, to the control panel to change control panel system settings for given controllers (see step **46** in FIG. **6**). Those skilled in the art will appreciate that the present invention is not limited to these approaches to applying the schemes to the control panel. These approaches are merely illustrative.

In a first approach, menu **66** (FIG. **8**A) is used to apply a scheme to the control panel. Menu **66** is provided by a controller when opened. Menu **66** includes a schemes option **68** that, when selected by the user, displays a list of schemes **70**. The user then selects one of the schemes to apply the settings of the scheme as the current settings for the controller.

A second approach that may be used to apply a scheme to the control panel is to activate a context menu **74** (FIG. **8**B) for the scheme. The context menu **74** is activated by clicking on the icon of the scheme using mouse **14**. The context menu **74** includes an option **76** to apply the scheme to the control panel. When the user selects this option, the scheme is applied and the system settings are changed in the control panel.

A third option for applying a scheme to the control panel is to perform a drag-and-drop operation. FIG. **8**C shows the steps that are performed in such a drag-and-drop operation for a scheme. First, a representation of the scheme, such as an open window or an icon, is dragged to a representation of the control panel (step **80**). The representation of the scheme is then dropped to apply the scheme to the control panel (step **82**).

It should be appreciated that each controller decides which settings may be encapsulated into a scheme. A controller may have some settings that are not encapsulated into a scheme. For example, system environment variables, such as the path to the system root, may be sorted with a controller but may not be part of a scheme. In addition, a scheme may include settings that are not applied to the control panel when the scheme is applied to the control panel. The settings that are not subject to being incorporated into schemes and the settings within schemes that are not subject to being applied to the control panel are disabled when the scheme is opened.

The above discussion notes that schemes or grand schemes may be dragged and dropped to realize the changing of system settings. The Microsoft OLE 2.0 protocol provides a number of interfaces that facilitate such drag-and-drop operations. An application registers a window as a drop target by calling a RegisterDragDrop( ) function that is provided by Microsoft OLE 2.0. In addition, the drop target supports the IDropTarget interface as defined within Microsoft OLE 2.0 and the drop source supports the IDropSource interface. More details regarding these interfaces can be found in *Inside OLE* 2, by Kraig Brockschmidt, Microsoft Press, 1994.

The preferred embodiment of the present invention provides a number of different ways for the user to create a scheme. FIG. **9**A is a flow chart showing the steps performed in one approach for creating a scheme. Initially, the user establishes the settings to be incorporated into a scheme using the dialog box **30** (FIG. **3**) of a controller (step **84**). The settings are then saved as a scheme (step **86**). For example, the background settings of no pattern and no wallpaper shown in FIG. **3** may be encapsulated into a scheme. The settings may be saved as a scheme by invoking option **72** within the menu **66** of the controller, as shown in FIG. **8**A. Option **72** saves the current settings of a controller as a scheme.

A second approach to creating a scheme is to perform a drag-and-drop operation. FIG. **9**B is a flow chart illustrating the steps that are performed to create a scheme in this fashion. A representation of a controller or scheme (such as an icon or a window for a controller or a scheme) is dragged using mouse **14** to a representation of a target scheme (step **88**). The representation of the controller or scheme is then dropped (step **90**). The target scheme then updates the settings encapsulated within it to reflect the settings of the controller or scheme that has been dropped upon it. In the Microsoft OLE 2.0 protocol, the target scheme is given a data pointer to a data object that holds the values of the settings for the controller or scheme that is dropped on it (step **92**). It accesses this data object to update its own settings. For example, suppose a user wishes to create a scheme holding current desktop settings. The user may then drag the desktop controller **28**D (FIG. **2**) and drop it in the scheme folder **36** (FIG. **3**) to create a scheme object holding the same desktop settings.

FIG. **9**C is a flow chart showing the steps that are performed in a third approach to create a scheme object. First, an existing scheme is copied by choosing an option such as the "Copy" option **77** in context menu **74** for a scheme (step **94**). The contents of the scheme are then modified to establish new settings and, thus, create the new scheme (step **96**).

FIG. **10** is a flow chart illustrating the steps the are performed to modify the settings within a scheme, such as in step **96** of FIG. **9**C. First, the scheme is opened so that a window is displayed and the contents of the scheme displayed (step **98**). A scheme may be opened by double clicking with the mouse **14** on an icon representing the scheme or by choosing the "open" option **79** (FIG. **8**B) from the context menu **74** of the scheme (step **98** in FIG. **10**). The contents of the scheme are then available and may be directly manipulated via dialog box **30**, which is the same dialog box that is used by the corresponding controller (FIG. **3**). The user may then modify the settings of the scheme using dialog box **30** (step **100** in FIG. **10**). The modified scheme is saved by choosing the "OK" button that is available within the dialog box **30** (step **102**).

Just as with schemes, there are a number of different approaches provided by the preferred embodiment of the present invention for modifying the contents of grand schemes and for creating grand scheme objects. FIG. **11**A is a flow chart illustrating a first approach to creating a grand scheme object. In accordance with this first approach, the user has opened the control panel and has selected option **56** (save settings as grand scheme) from the control panel menu **50** (step **104**). This option **56** requests the user to select which controllers are to have their settings incorporated into the grand scheme. The user selects the controller using the dialog provided by option **56** (step **106** in FIG. **11**A).

FIG. **11**B is a flow chart showing the steps that are performed in a second approach to create a grand scheme object. First, an existing grand scheme is copied, such as in

response to a user selecting copy option **61** (FIG. **7**B) from the grand scheme context menu **58** (step **108** in FIG. **11**B). The user then may modify the settings within the grand scheme by opening each scheme contained therein and editing the contents (step **110**).

FIG. **11**C shows a flow chart of the steps that are performed in an option for creating an object holding system settings. First, a representation of a control panel is dragged to a representation of a folder, such as a schemes folder (step **112**). The representation is then dropped (step **114**), causing a new object holding the system settings of the control panel to be added to the folder (step **116**).

FIG. **12** shows an approach for modifying the contents of a grand scheme. In this approach, a controller or scheme is dragged or dropped onto a grand scheme. In step **126**, a representation of a controller or scheme is dragged to a representation of a grand scheme. The representation of the controller scheme is then dropped (step **128**). The settings for the scheme that corresponds with the controller or scheme that is dropped are then updated to reflect the values of the controller or scheme that has been dropped (step **130**). Those skilled in the art will appreciate that a grand scheme may be directly modified by opening its controllers and modifying the system settings.

While the present invention has been described with reference to a preferred embodiment thereof, those skilled in the art will nevertheless appreciate that various changes and forms in detail may be made without departing from the intended scope of the present invention as defined in the appended claims.

What is claimed is:

1. In a computer system having a video display and a storage device and running an operating system that provides a desktop environment to a user and a file system, said desktop environment having associated system settings that affect the desktop environment, a method comprising the steps of:

storing in the storage device a first set of values for at least a portion of the system settings so that the first set of values is visible in the file system;

storing in the storage device a second set of values for the same portion of the system settings for which values are stored in the first set of values so that the second set of values is visible in the file system; and

in response to a user choice between the first set of values and the second set of values, updating the systems settings to have the values of the set of values that has been chosen by the user.

2. The method of claim **1** wherein the step of storing in the storage device the first set of values comprises the step of storing in the storage device the first set of values for all of the system settings that a user may change so that the first set of values is visible in the file system.

3. The method of claim **1** wherein at least one of the system settings affects appearance of the desktop environment.

4. The method of claim **1** wherein at least one of the system settings affects behavior of the desktop environment.

5. The method of claim **1** wherein the first set of values and the second set of values are stored as a first object and second object, respectively.

6. The method of claim **1**, further comprising the step of storing in the storage device a third set of values for the same portion of the system settings for which values are stored in the first and second sets of values, said third set of values being visible in the file system.

7. In a computer system having a storage device and running an operating system that provides an operating environment to a user, said operating environment being specified by values of system settings, a method comprising the steps of:

providing a control panel for controlling current values of the system settings;

storing sets of values for the system settings in the storage device, each set including values for at least a portion of the system settings;

providing an interface for enabling the user to select one of the sets of values; and

in response to a selection of one of the sets of values by the user, changing the current system settings to have the values of the selected set of values.

8. The method of claim **7** wherein the set of values selected by the user includes values for all of the system settings whose values are controlled by the control panel.

9. The method of claim **7** wherein the set of values selected by the user includes values for only a subset of the system settings whose values are controlled by the control panel.

10. The method of claim **7**, further comprising the step of providing a user interface for the control panel that enables the user to select at least one new value for the system settings.

11. In a computer system running an operating system that provides an operating environment to a user and a file system wherein the operating environment has system settings that affect the operating environment, a method comprising the steps of:

providing a control panel for controlling current values of the system settings, the control panel including controllers that each are responsible for controlling current values of a group of related ones of the system settings;

providing the user with at least two sets of values for an identified one of the groups of related system settings such that the sets of values are visible in the file system; and

in response to a selection by the user of one of the sets of values, changing the current values of the identified group of related system settings to the values in the set of values selected by the user.

12. The method of claimed **11**, further comprising the step of providing the user with additional sets of values visible in the file system for ones of the groups of related system settings that differ from the identified group.

13. The method of claim **12**, further comprising the step of, in response to selection by the user of certain of the additional sets of values, changing the system settings for the groups of related system settings for which the selected additional sets of values hold values, to the values held in the selected additional sets of values.

14. The method of claim **11**, further comprising the step of providing the user with another set of values visible in the file system for system settings in multiple ones of the groups of related system settings.

15. The method of claim **14**, further comprising the step of, in response to the user selecting the other set of values, changing the current values of the multiple groups for which the other set of values holds values, to the values in the other set.

16. In a computer system having a storage device and running an operating system that provides an operating environment to a user, said operating environment conforming to current system settings, a method comprising the steps of:

storing a grand scheme container object in the storage device;

storing scheme objects within the grand scheme container object, each scheme object holding a set of values for a subset of the system settings;

applying the values held in the scheme objects contained in the grand scheme container object to the current system setting so as to change the values of the current system settings to the values held in the scheme objects of the grand scheme container object.

17. The method of claim **16**, further comprising the step of storing an additional grand scheme container object holding scheme objects in the storage device.

18. The method of claim **17**, further comprising the step of storing the grand scheme container objects and the scheme objects in a folder object.

19. In a computer system having an input device and a video display and running an operating system that provides an operating environment to a user as specified by system settings, a method comprising the steps of:

providing a first object holding values for system settings, said first object having a representation on the video display;

providing a second object holding system settings, said second object having a representation on the video display;

dragging the representation of the first object on the video display to lie over at least a portion of the representation of the second object in response to the user using the input device;

dropping the representation of the first object on the representation of the second object in response to the user using the input device;

in response to the dropping of the representation of the first object on the representation of the second object, changing the values for system settings in the second object to the values for system settings in the first object.

20. The method of claim **19** wherein the second object is a control panel object that controls current values for the system settings.

21. The method of claim **20** wherein the first object is a scheme object holding values for only a subset of the system settings whose values that are controlled by the control panel, thereby changing only the subset.

22. The method of claim **20** wherein the first object is a grand scheme object holding values for all of the system settings whose values are controlled by the control panel, thereby changing all of the values.

23. The method of claim **19** wherein the second object is a grand scheme object containing scheme objects that each hold values for subsets of the system settings whose values are controlled by the control panel and the first object is an additional scheme object that holds values for a subset of the system settings whose values are controlled by the control panel.

24. The method of claim **19** wherein the first object is a grand scheme object and the second object is a control panel object that controls current values for the system settings.

25. The method of claim **19** wherein both the first object and the second object are scheme objects.

26. A data processing system comprising:

a processor for running an operating system that provides a file system and a desktop environment to the user, said desktop environment having an associated set of system settings that affect the desktop environment;

a storage comprising:

(i) a copy of the operating system;

(ii) a first set of values for at least a portion of the system settings so that the first set of values is visible in the file system;

(iii) a second set of values for the same portion of the system settings for which values are stored in the first set of values so that the second set of values is visible in the file system; and

a vehicle for updating the system settings, in response to a user choice of one of the sets of values, to have the values of the set of values that was chosen by the user.

27. The data processing system of claim **26**, further comprising a folder in the storage that holds a first set of values and the second set of values.

28. A system for providing a desktop environment to a user, said desktop environment having associated system settings that affect it, comprising:

a display component for displaying an interface to a user as part of the desktop environment according to the system settings;

a first container holding a first set of system settings values;

a second container holding a second set of system settings values;

a selection component that receives selection information and in response selects between the first container and the second container; and

a change component responsive to the selection component for changing the system settings to those held in the selected container.

29. A computer-readable storage medium for use in a computer system having a video display and a storage device and running an operating system that provides a desktop environment to a user and a file system, wherein said desktop environment has associated system settings that affect the desktop environment, said medium holding:

a first set of values for at least a portion of the system settings wherein the first set of values is visible in the file system;

a second set of values for the same portion of the system settings for which values are stored in the first set of values wherein the second set of values is visible in the file system; and

a system settings update component for updating the system settings to have the values of a one of the first set of values, the second set of values, or that has been chosen by a user.

30. The computer-readable storage medium of claim **29** wherein the first set of values is stored in a first object and the second set of values is stored in a second object.

31. A computer-readable storage medium for use in a computer system having a storage device and running an operating system that provides an operating environment to a user that conforms to current system settings, said medium holding:

a grand scheme container object that holds scheme objects wherein each scheme object holds a set of values for a subset of the system settings; and

a component for applying the values held in the scheme objects in the grand scheme container object to the current system settings so as to change the values of the current system settings to the values held in the scheme objects of the grand scheme container object.

32. A computer-implemented method for modifying an appearance of a desktop environment of a computer system,

the desktop environment providing a plurality of parameters, each parameter settable to values that control the appearance of an aspect of the desktop environment, the method comprising:

receiving from a user a first set of values for the parameters;

storing the received first set of values;

receiving from the user a second set of values for the parameters;

storing the received second set of values;

receiving from the user a selection of either the first set of values or the second set of values;

when the selection of the first set of values is received from the user, setting the parameters to the values of first set of values; and

when the selection of the second set of values is received from the user, setting the parameters to the values of the second set of values

whereby the user can specify sets of values and whereby the computer system sets the parameters to the values of a set when the user selects that set of values so that the appearance of the desktop environment is modified in accordance with the selected set of values.

33. The method of claim **32** wherein the storing of the received sets of values stores the sets of values using a file system.

34. The method of claim **32** wherein the user selects a set of values by dragging and dropping a visual representation of the set of values over a visual representation of the parameters.

35. The method of claim **32** wherein the parameters are set under control of an operating system.

36. The method of claim **32** including changing a value in the first set of values so that subsequent selection of the first set results in modifying the appearance of the desktop environment in accordance with the changed value.

37. The method of claim **32** wherein the parameters are divided into control groups, each control group having a controller for setting the values of the parameters of that control group, and wherein the first set of values is for parameters of a plurality of control groups.

38. The method of claim **37** wherein the setting of the values of the parameters is performed by the controllers for the plurality of control groups.

39. A computer-readable medium containing instructions for causing a computer system to modify the appearance of a desktop environment of a computer system, the desktop environment providing a plurality of control groups having parameters, each parameter of a control group settable to a value that controls the appearance of an aspect of the desktop environment, each control group having a controller for controlling the setting of the values of the parameters in the control group, by:

storing in a file a first set of values for parameters included in more than one control group;

storing in a file a second set of values for the parameters included in more than one control group;

receiving from a user a selection of either the stored first set of values or the stored second set of values;

when the selection of the first set of values is received from the user, invoking the controllers of the control groups that include parameters of the first set of values to set the parameters to the values of first set of values; and

when the selection of the second set of values is received from the user, invoking the controllers of the control groups that include parameters of the second set of values to set the parameters to the values of second set of values

whereby the user can select a set of values so that the values of the parameters of more than one control group can be set by the selection of the set of values.

40. The computer-readable medium of claim **39** wherein the user selects a set of values by dragging and dropping a visual representation of the set of values over a visual representation of the parameters.

41. The computer-readable medium of claim **39** wherein the parameters are set under control of an operating system.

42. The computer-readable medium of claim **39** including changing a value in the first set of values so that subsequent selection of the first set results in modifying the appearance of the desktop environment in accordance with the changed value.

* * * * *

Exhibit F

US006542164B2

(12) **United States Patent**
Graham

(10) **Patent No.: US 6,542,164 B2**
(45) **Date of Patent: *Apr. 1, 2003**

(54) **TIMING AND VELOCITY CONTROL FOR DISPLAYING GRAPHICAL INFORMATION**

(75) Inventor: **Christopher E. Graham**, Redmond, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/879,479**

(22) Filed: **Jun. 12, 2001**

(65) **Prior Publication Data**

US 2002/0054013 A1 May 9, 2002

Related U.S. Application Data

(63) Continuation of application No. 08/873,855, filed on Jun. 12, 1997, now Pat. No. 6,281,879, which is a continuation of application No. 08/709,529, filed on Sep. 6, 1996, now abandoned, which is a continuation of application No. 08/260,558, filed on Jun. 16, 1994, now abandoned.

(51) **Int. Cl.**[7] **G09G 5/00**
(52) **U.S. Cl.** **345/711**; 345/157
(58) **Field of Search** 345/157, 159, 345/156, 160, 856, 859, 711, 715

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,789,962 A		12/1988	Berry et al.	364/900
4,984,152 A		1/1991	Muller	364/200
5,140,678 A	*	8/1992	Torres	
5,155,806 A		10/1992	Hoeber et al.	395/157
5,157,768 A		10/1992	Hoeber et al.	395/157
5,169,342 A		12/1992	Steele et al.	434/112
5,196,838 A		3/1993	Meier et al.	340/724
5,287,448 A	*	2/1994	Nicol	
5,299,307 A	*	3/1994	Young	345/157
5,546,521 A	*	8/1996	Martinez	

OTHER PUBLICATIONS

Macintosh Reference, Apple Computer Inc, pp. 30–31, 1991.*
Quick Result, Microsoft Word, Version 6.0, 1993, pp. 39–40 and 154–155.*

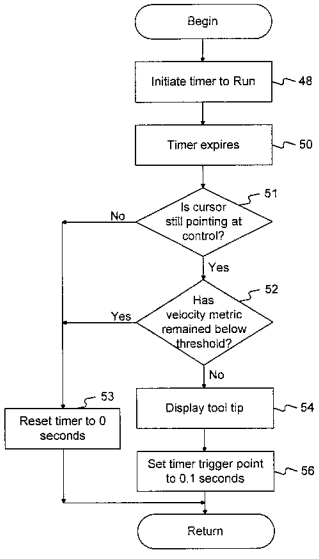
* cited by examiner

Primary Examiner—Chanh Nguyen
(74) *Attorney, Agent, or Firm*—Merchant & Gould P.C.

(57) **ABSTRACT**

Time and velocity metrics are used to control when information about a graphical object to which a cursor points is displayed on a video display. The time metric is used to ensure that a non-negligible amount of time passes between the time at which the cursor initially points to the graphical object and the time at which the information about the graphical object is displayed on the video display. The time delay helps to eliminate such information being displayed inadvertently when the user quickly passes the cursor over graphical objects in the video display. In addition, the timing control facilitates the shortening of the delay when it appears that the user wishes to browse amongst several related graphical objects that are shown in the video display. For example, when it appears that the user wishes to browse tools on the tool bar, the delay is shortened. The velocity metric is used to determine the likelihood that the user intended to point to the graphical object and serves to minimize instances where undesired information about the graphical object is displayed.

30 Claims, 7 Drawing Sheets



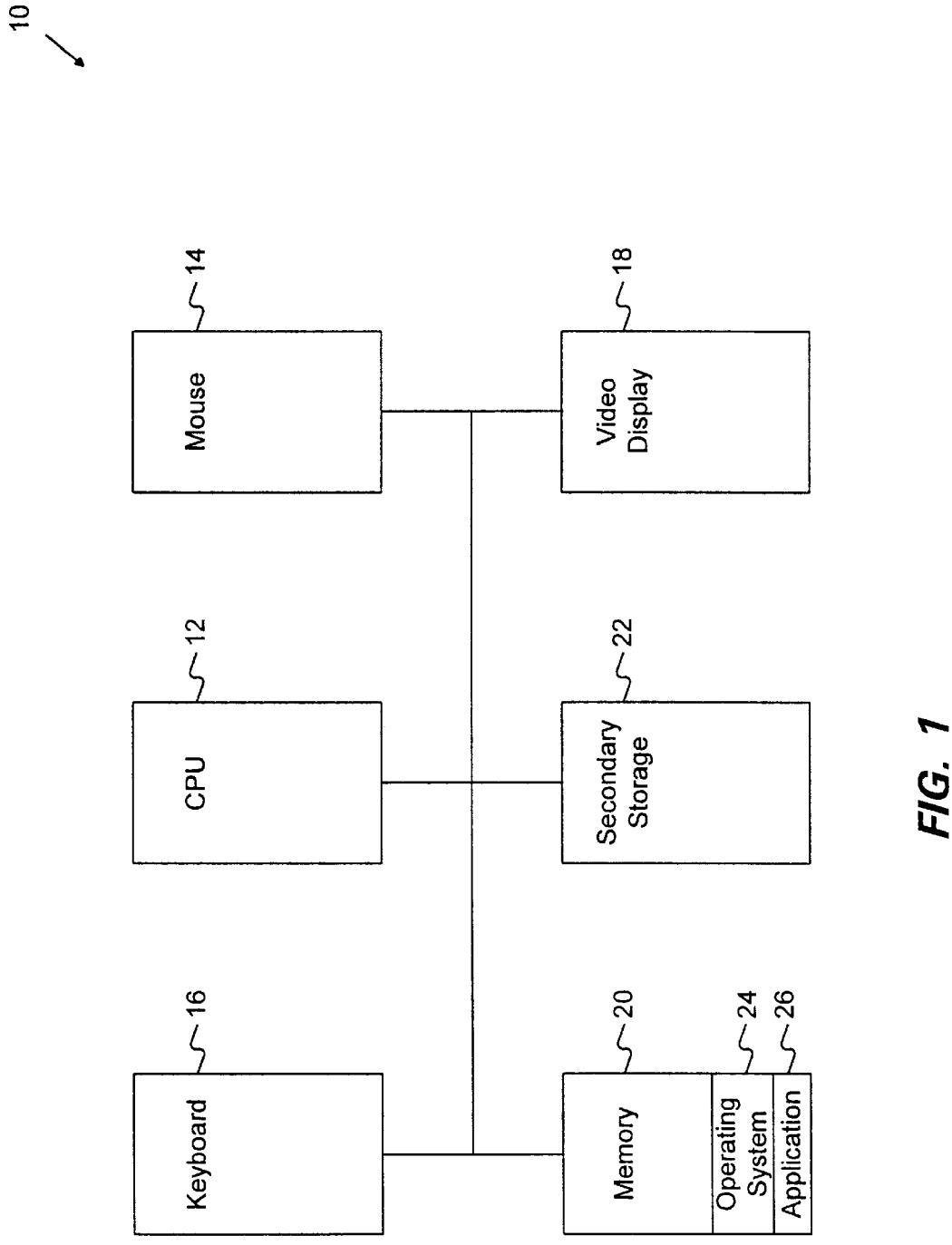

10
14
Mouse
18
Video Display
12
CPU
22
Secondary Storage
16
Keyboard
20
Memory
24
Operating System
26
Application

FIG. 1

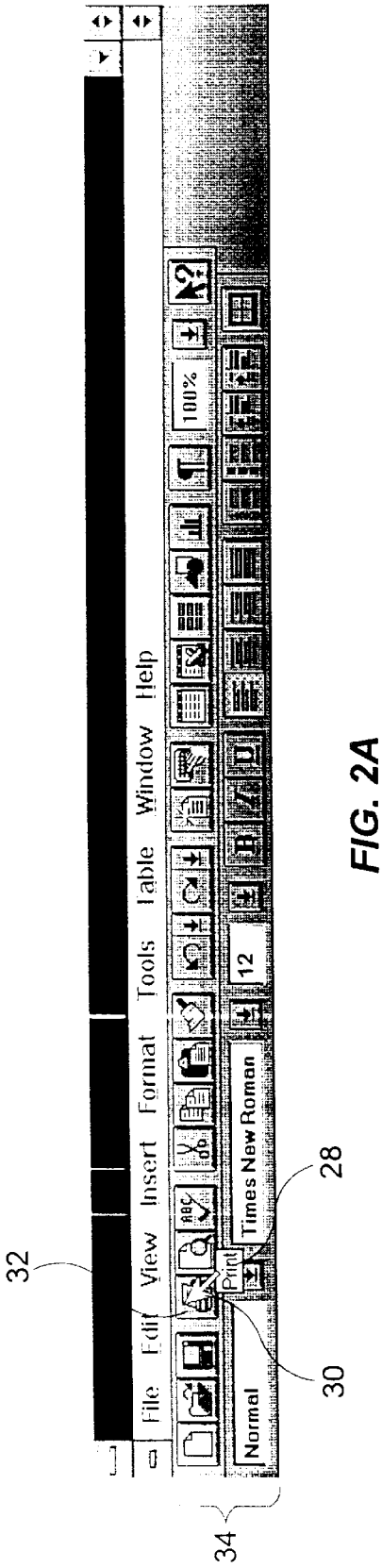

File Edit View Insert Format Tools Table Window Help
Normal
Times New Roman
12
100%
Print
32
28
30
34

FIG. 2A

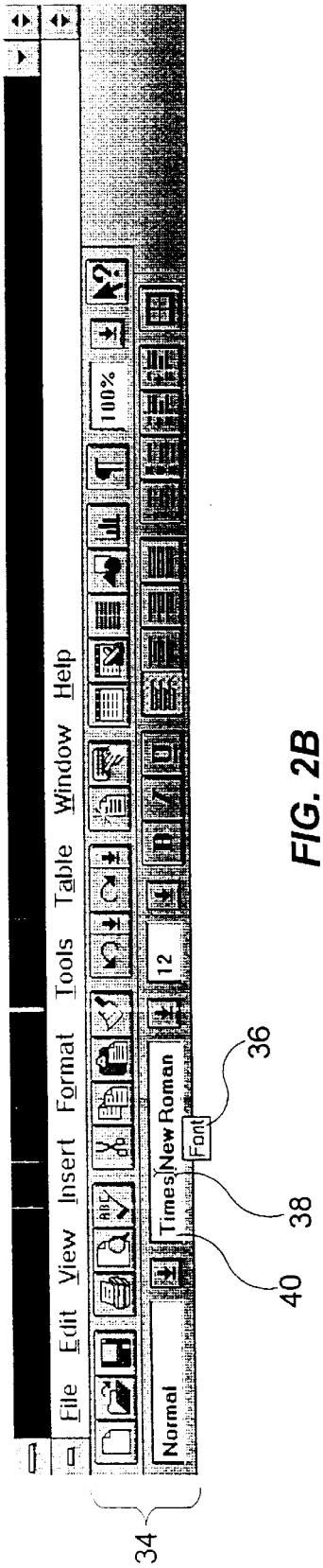

File Edit View Insert Format Tools Table Window Help
Normal
Times New Roman
12
100%
Font
36
38
40
34

FIG. 2B

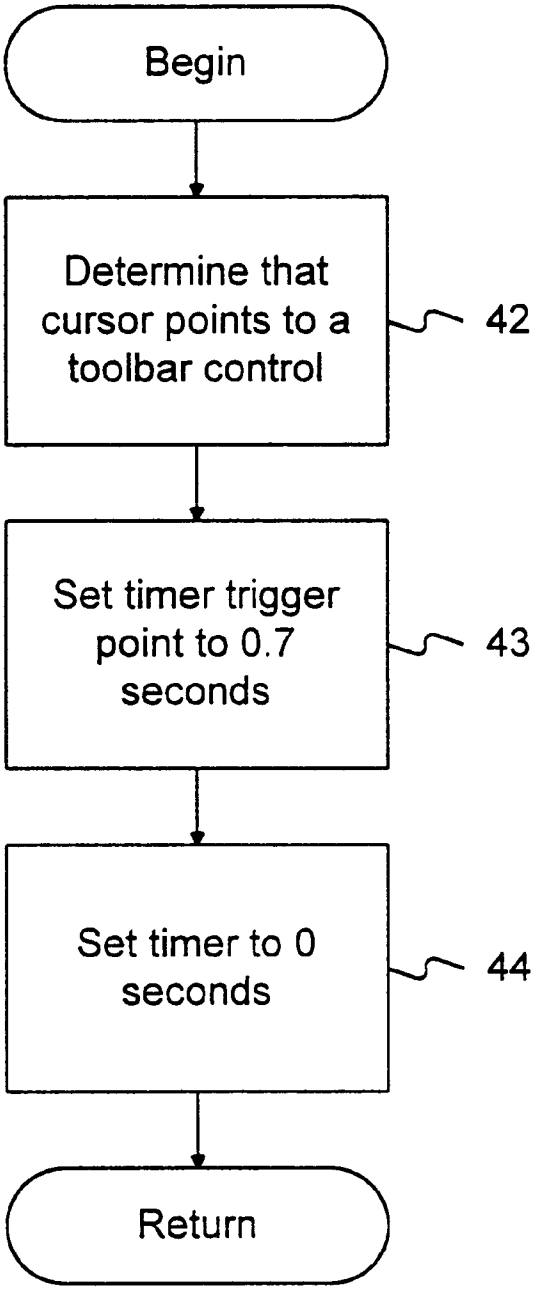

Begin
Determine that cursor points to a toolbar control
42
Set timer trigger point to 0.7 seconds
43
Set timer to 0 seconds
44
Return

FIG. 3A

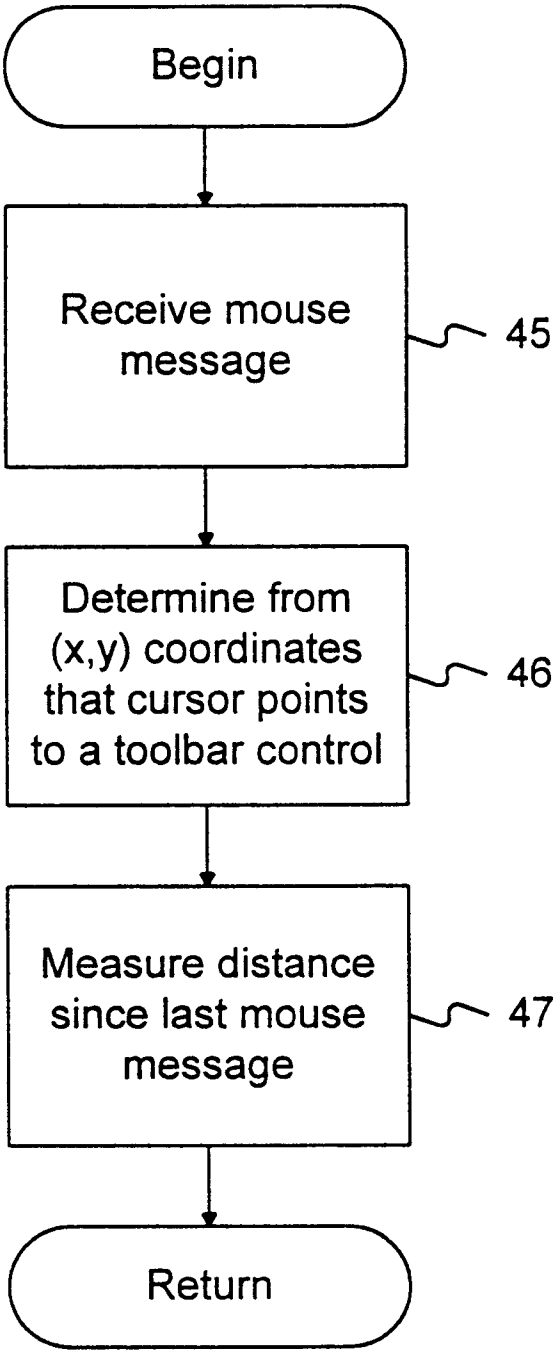

Begin
Receive mouse message
45
Determine from (x,y) coordinates that cursor points to a toolbar control
46
Measure distance since last mouse message
47
Return

FIG. 3B

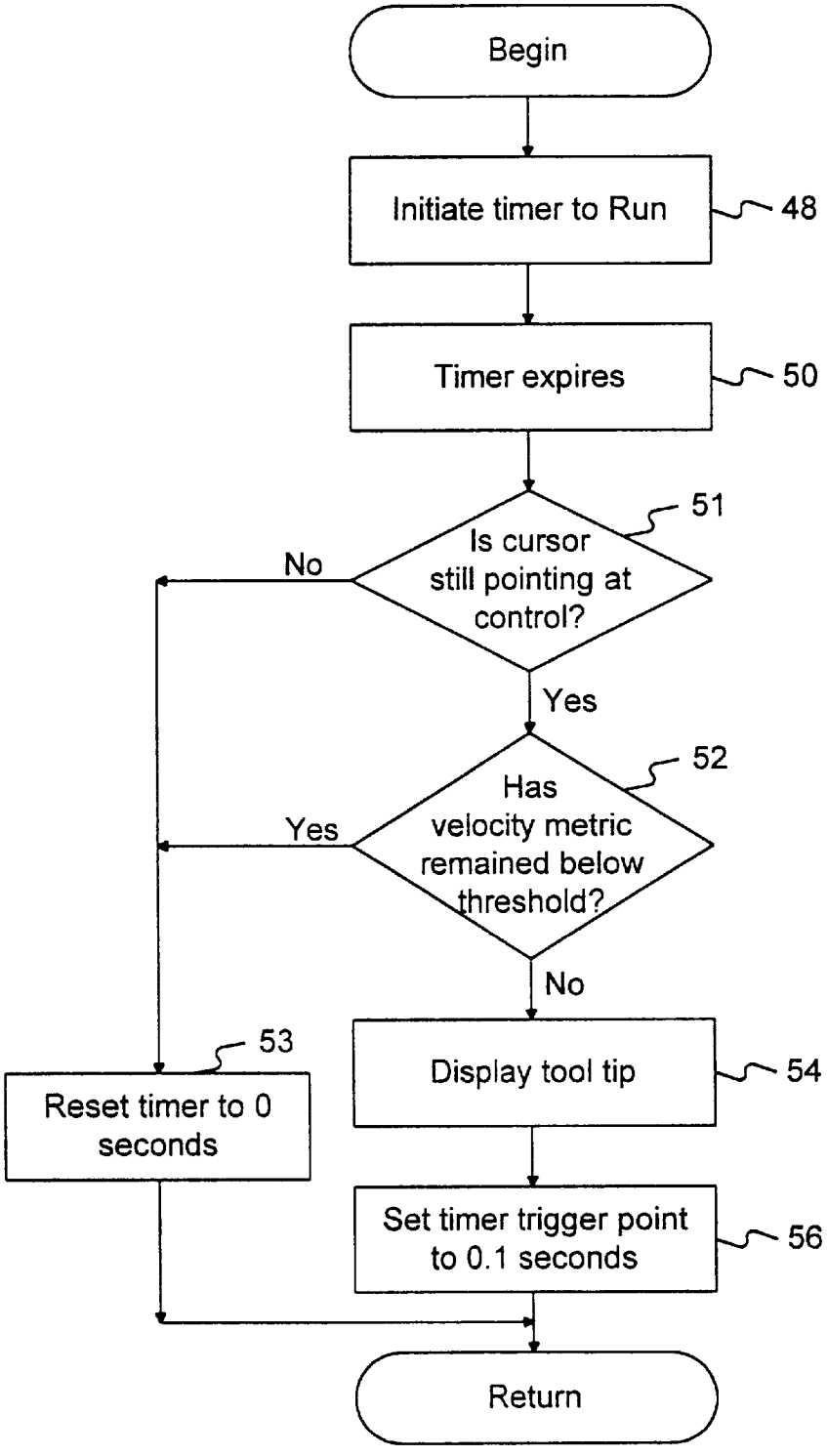

Begin
Initiate timer to Run
48
Timer expires
50
51
Is cursor still pointing at control?
No
Yes
52
Has velocity metric remained below threshold?
Yes
No
53
Reset timer to 0 seconds
Display tool tip
54
Set timer trigger point to 0.1 seconds
56
Return

FIG. 4

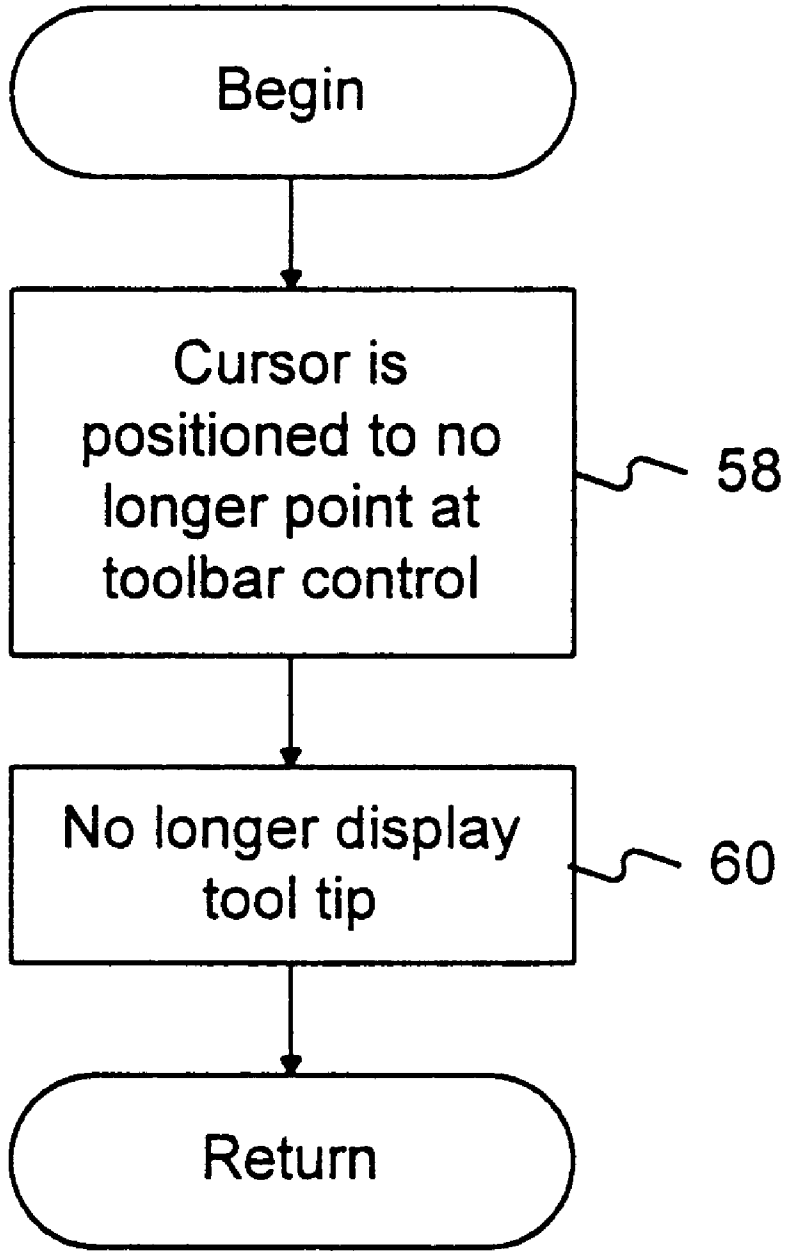

Begin
Cursor is positioned to no longer point at toolbar control
58
No longer display tool tip
60
Return

FIG. 5

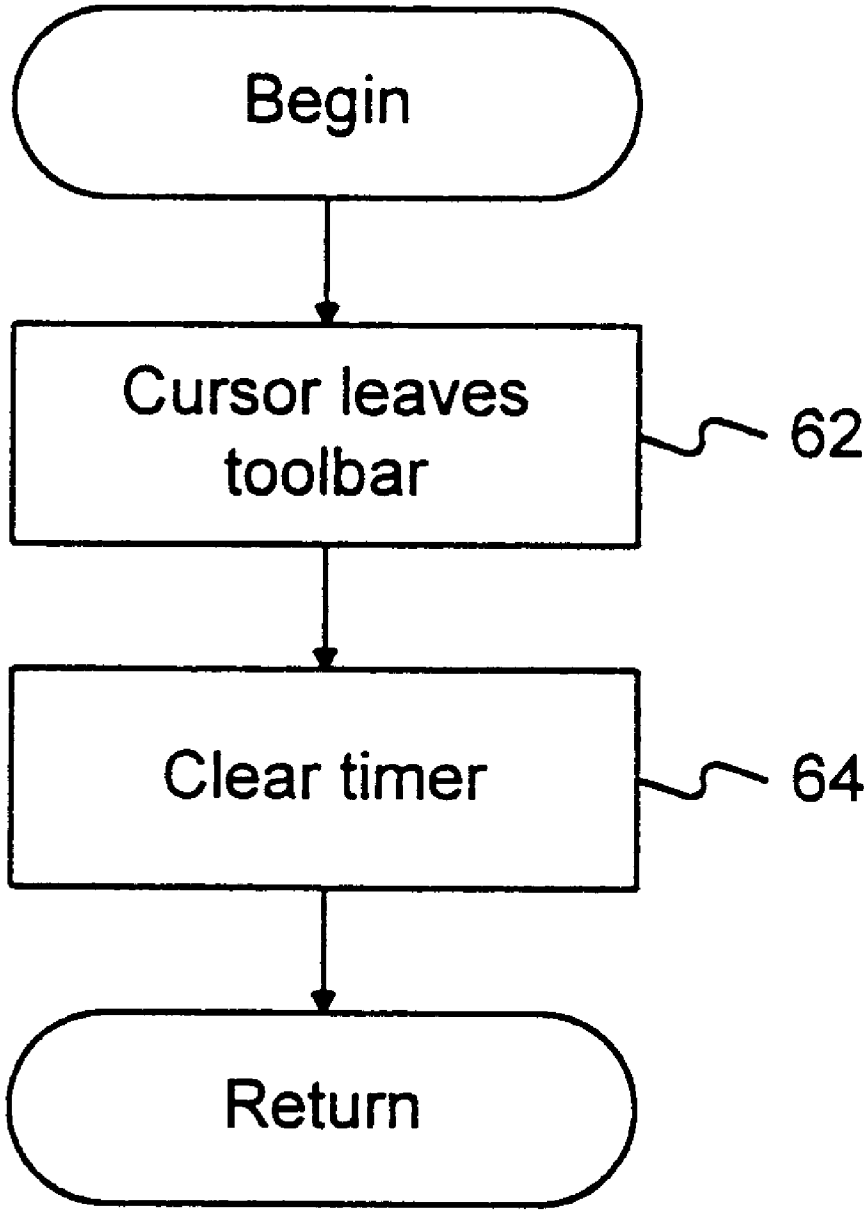


FIG. 6

TIMING AND VELOCITY CONTROL FOR DISPLAYING GRAPHICAL INFORMATION

RELATED APPLICATIONS

This is a continuation of U.S. patent application Ser. No. 08/873,855, filed Jun. 12, 1997, now U.S. Pat. No. 6,281,879, which is a continuation of U.S. patent application Ser. No. 08/709,529, filed Sep. 6, 1996, now abandoned, which is a File Wrapper Continuation of U.S. patent application Ser. No. 08/260,558, filed Jun. 16, 1994, now abandoned, which applications are incorporated herein by reference.

TECHNICAL FIELD

The present invention relates generally to data processing systems and, more particularly, to the displaying of graphical information in data processing systems.

BACKGROUND OF THE INVENTION

Many conventional application programs utilize tool bars. Tool bars provide the user with a number of tools that assist the user in performing tasks. Typically, a separate control is provided for each tool on the tool bar. The control may be a pushbutton or another graphical object that allows the user to invoke the desired tool. Often times the controls on the tool bar have icons on their faces that indicate the nature of the tool. Unfortunately, as the typical number of controls on the tool bar has grown for applications, it has become more and more difficult for the user to discern the nature of the tool solely from the icons shown as part of the tool bar. As such, many users have difficulty using the tools on the tool bar.

SUMMARY OF THE INVENTION

The limitations of the prior art are overcome by the present invention. In accordance with a first aspect of the present invention, a method is practiced in a data processing system having a video display for displaying a cursor that points to positions on the video display. The data processing system also includes an input device for manipulating the cursor. In accordance with this method, it is first determined that the cursor points to a position within a region on the video display. A velocity metric of the cursor is measured. Where the velocity metric does not exceed a predetermined threshold value, an event is triggered. On the other hand, where the velocity metric exceeds the predetermined threshold value, the event is inhibited.

In accordance with a second aspect of the present invention, it is determined that a cursor points to a position within a region on the video display. A time period metric that specifies how long the cursor has remained pointing within the region is measured. A velocity metric of the cursor within the region is also measured. Based upon these metrics, a determination is made whether to trigger an event.

In accordance with an additional aspect of the present invention, a method is practiced in a data processing system having a video display for displaying a cursor that points to positions in the video display and an input device for manipulating the cursor. In accordance with this method, a graphical object is displayed on the video display. The user uses the input device, and in response, the data processing system positions the cursor to point at the graphical object. A predetermined period of time, such as a time greater than 0.4 seconds, is allowed to pass and then a determination is made whether the cursor still points at the graphical object. If it is determined that the cursor still points at the graphical object, information about the graphical object is displayed adjacent to the graphical object on the video display.

In accordance with another aspect of the present invention, a method is practiced wherein a tool bar having tools is displayed on the video display. When the user uses the input device, the cursor is position to point at a selected one of the tools on the tool bar. The system waits a predetermined non-negligible amount of time. The system also measures a velocity metric of the cursor within the first graphical object. If the cursor still points at the selected tool after waiting the predetermined non-negligible amount of time and the velocity metric has remained below a predetermined threshold during the predetermined non-negligible amount of time, information about the selected tool is displayed in the video display. The position is adjacent to the selected tool.

In accordance with a further aspect of the present invention, a method is practiced in a computer system having a video display for displaying a cursor that points to positions on the video display and an input device for moving the cursor on the video display. In this method, a first graphical object is displayed on the video display. In response to the user using the input device, the cursor is positioned to point at the first graphical object. The system waits a non-negligible predetermined amount of time. A determination is made whether the cursor still points at the graphical object after the non-negligible predetermined amount of time has passed. Where the cursor still points at the first graphical object, a number of steps are performed. These steps include displaying information about the first graphical object adjacent to the first graphical object of the video display. The non-negligible predetermined amount of time is then reset to a substantially shorter amount of time. A second graphical object is displayed on the video display and, in response to the user using the input device, the cursor is positioned to point at the second graphical object on the video display. The system waits the substantially shorter amount of time. Where the cursor is still pointing at the second graphical object after waiting the substantially shorter period of time, information about the second graphical object is displayed adjacent to the second graphical object on the video display.

In accordance with a still further aspect of the present invention, a data processing system includes a video display for displaying video data. The video display displays a first graphical object and a cursor that points to the first graphical object. An input device is included in this part of the data processing system for moving the cursor on the video display. A message generator is provided for displaying information about the first graphical object. The information is displayed adjacent to the first graphical object on the video display when the cursor remains pointing at the first graphical object for a predetermined non-negligible amount of time. The message generator includes a comparator and a message source. The comparator determines whether the cursors remain pointing at the first graphical object for the predetermined non-negligible amount of time. The message source provides and displays information about the first graphical object adjacent to the first graphical object on the video display when the comparator determines that the cursor has remained pointing at the first graphical object for the specified amount of time.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a data processing system that is suitable for practicing a preferred embodiment of the present invention.

FIG. **2A** is a diagram illustrating a tool bar and a tool tip that is provided for a print button in accordance with the preferred embodiment of the present invention.

FIG. **2B** is a diagram illustrating a tool bar and a tool tip that is provided for a font list box in accordance with the preferred embodiment of the present invention.

FIG. **3A** is a flow chart illustrating the steps performed to initially set a timer when a cursor is positioned over a control on the tool bar in accordance with the preferred embodiment of the present invention.

FIG. **3B** is a flow chart illustrating the steps performed to determine the magnitude of the velocity of the cursor when the cursor is positioned over a control on the tool bar in accordance with the preferred embodiment of the present invention.

FIG. **4** is a flow chart illustrating the steps performed to determine whether a tool tip is to be displayed in the preferred embodiment of the present invention.

FIG. **5** is a flow chart illustrating the steps performed when a cursor no longer points within the tool bar in the preferred embodiment of the present invention.

FIG. **6** is a low chart illustrating the steps that are performed relative to the timer when the cursor leaves the tool bar in the preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The preferred embodiment of the present invention displays a tool tip when a mouse cursor points to a tool or a tool bar for a sufficient amount of time and the magnitude of the velocity of the mouse cursor remains below a predetermined threshold. A tool tip is a brief textual message, such as a name of a tool, that identifies the nature of the tool. The preferred embodiment of the present invention provides a delay between when the mouse cursor is initially positioned to point at the tool control on the tool bar and when the tool tip is displayed. This delay prevents the user from receiving undesired tool tips when the user inadvertently passes the mouse cursor over a control on the tool bar. The delay is sufficiently long (i.e., it is non-negligible) to allow the user to move the mouse cursor if he does not want to receive a tool tip. The delay is shortened when an initial tool tip is displayed so as to allow the user to quickly browse the tool controls that are available on the tool bar. The magnitude of the velocity of the mouse cursor is measured to determine whether the user likely intends to point at the tool to receive a tool tip or whether the user, instead, is merely passing over the tool while moving to another destination.

Although the preferred embodiment of the present invention concerns controlling when tool tips for tools on a tool bar are displayed, those skilled in the art will appreciate that the present invention may generally be applied to other regions of a video display (such as other types of graphical objects) for which graphical information is to be provided. It should be appreciated that the present invention may be applied to both visible and invisible graphical objects.

FIG. **1** is a block diagram of a data processing system **10** that is suitable for practicing the preferred embodiment of the present invention. The data processing system **10** includes at least one central processing unit (CPU) **12**. The CPU **12** is connected to a number of peripheral devices, including a mouse **14**, a keyboard **16** and a video display **18**. The CPU **12** is also connected to a memory **20** and a secondary storage device **22**, such as a hard disk drive. The memory **20** holds a copy of an operating system **24**, such as the Microsoft Windows, version 3.1, operating system sold by Microsoft Corporation of Redmond, Wash. The memory **20** also holds a copy of an application program **26**. The implementation of the preferred embodiment of the present invention will be described below with reference to use of tool tips within the application program **26**. Nevertheless, it should be appreciated that the tool tips may alternatively be implemented in the operating system **24** or as a system resource.

FIG. **2A** shows the tool bar **34** that is generated and displayed on the video display **18** when the application program **26** is run on the CPU **12**. The tool bar **34** includes a number of controls, such as buttons and list boxes, that enable a user to access the tools of the tool bar. When the user positions a mouse cursor **30** over one of the buttons and clicks the mouse (i.e., quickly depresses and releases a predefined one of the mouse buttons), the tool associated with the button is invoked. Similarly, by positioning the mouse cursor **30** over one of the buttons of the list boxes, a drop-down list appears, and the user may select one of the options on the drop-down list using the mouse **14**.

The tool bar **34** is created as a window by the application program **26**. The operating system **24** facilitates the definition of such windows (as is provided in the Microsoft Windows, version 3.1, operating system). FIG. **2A** shows the mouse cursor **30** pointing to a print button **32** on the tool bar **34**. When a user positions the mouse cursor **30** over the print button **32** and clicks the mouse button, the document currently displayed in the window of the application program is printed.

The user interface provided for the application program **24** is logically divisible into a number of windows. One of these windows is the tool bar **34**. In general, each window of the user interface has a separate window procedure associated with it. The operating system **24** maintains a message queue for each program that generates windows. Accordingly, the application program **26** has its own message queue. Since the application program **26** may generate multiple windows, the message queue may hold messages for multiple windows. When an event occurs, the event is translated into a message that is put into the message queue for the application program **26**. The application program **26** retrieves and delivers the message to the proper windows by executing a block code known as the "message loop." The window procedure that receives the message then processes the message.

Movements of the mouse **14** are reflected in messages that are placed into the message queue of the application program **26**. In particular, when a user positions the mouse cursor **30** with the mouse **14** over a window or clicks the mouse by depressing one of the mouse buttons within a window, the procedure for the window receives a mouse message. The operating system **24** provides a number of predefined mouse messages. The mouse messages specify the status of mouse buttons and the position of the mouse cursor **30** within the window. The position of the mouse cursor **30** within the window is specified in (X, Y) coordinates relative to the upper left-hand corner of the window. Thus, when the mouse cursor **30** moves within the tool bar **34**, the position of the mouse cursor **30** within the tool bar is reflected and a mouse message that specifies (X, Y) coordinates of the mouse cursor relative to the upper left-hand corner of the tool bar. The window procedure receives the mouse message and utilizes the information contained in the message to respond to the mouse **14** activities.

As mentioned above, the application program **26** specifies the window that constitutes the tool bar **34**. The application

program **26** paints each of the controls, including print button **32**, at known locations within the window of the tool bar **34**. When the mouse cursor **30** is positioned within the tool bar, mouse messages specify the position of the mouse cursor within the tool bar **34**. The mouse messages are sent to the window procedure that is responsible for the tool bar window. The window procedure for the tool bar **34** compares the coordinates specified by the mouse message with the known location of the controls within the tool bar. Accordingly, the window procedure for the tool bar **34** can determine whether the mouse cursor **30** is pointing at any of the controls. When it is determined that the mouse cursor **30** is pointing at one of the controls of the tool bar **34**, a tool tip **28** may be displayed if the mouse cursor **30** has remained pointing at the control for a sufficient period of time and the magnitude of the velocity of the mouse cursor is below a predetermined threshold. Hence, in the example shown in FIG. **2**A, the message "Print" is displayed as a tool tip **28**, given that the mouse cursor **30** is pointing to the print button **32**.

Tool tips are provided not only for buttons on the tool bar **34** but are also provided for other types of controls. For example, as shown in FIG. **2**B, a mouse cursor **38** points to a portion of a list box **40** that concerns the font which the user wishes to utilize. Accordingly, a tool tip **36** is displayed that includes the text "Font". It is worth noting that the mouse cursor **38** changes from an arrow to a cross bar, since the cursor points to a portion of a list box that contains text rather than a button as in FIG. **2**A.

Tool tips are displayed using text output commands that are provided by the operating system **24**. Specifically, the ExtTextOut( ) function that is provided by the Microsoft Windows, version 3.1, operating system is used in the preferred embodiment. The format of this function is as follows:

```
BOOL ExtTextOut(hdc, nXStart, nYStart, fuOptions, lprc, lpszString, cbString, lpDx)
HDC hdc;                /* handle of device context                */
int nXStart;            /* x-coordinate of starting position       */
int nYStart;            /* y-coordinate of starting position       */
UINT fuOptions;         /* rectangle type                          */
const RECT FAR* lprc;   /* address of structure with rectangle     */
LPCSTR lpszString;      /* address of string                       */
UINT cbString;          /* number of bytes in string               */
int FAR* lpDx;          /* spacing between character cells         */
```

The hdc parameter of this function specifies a handle (i.e., a numerical identifier) for a device context. In this case, the device context specifies attributes that determine how the operating system interacts with the video display **18**. The nXStart parameter specifies the logical X coordinate at which the string of the tool tip message begins. Similarly, the nYStart parameter specifies the logical Y coordinate at which the string begins. The fuOptions parameter specifies the type of rectangle for the tool tip. The operating system **24** provides predefined data structures that specify rectangle types. In this case, the rectangle type is defined as a clipped rectangle. The lprc parameter is a pointer to a structure that holds a rectangle and the lpszString is a pointer to a structure that holds the textual string to be displayed in the tool tip. The cbString parameter specifies the number of bytes in the string and the lpDx parameter specifies spacing between character cells.

When the window procedure for the tool bar **34** receives a mouse message that indicates that the mouse cursor **30** (see FIG. **2**A) is positioned over one of the controls of the tool bar **34**, a sufficient time has elapsed and the measured magnitude of the velocity of the mouse cursor **30** is below a predetermined threshold, the procedure determines the string that is to be displayed in the tool tip for the control to which the mouse cursor points. The address of this string is passed as the lpszString parameter to the ExtTextOut( ) function. This function then proceeds to draw the tool tip.

As the rectangle for the tool tip is a clipped rectangle, the background color may be specified. In the preferred embodiment of the present invention, the background color is yellow, as specified in red/green/blue (RG3) coordinates as (255, 255, 128). The size of the rectangle used for the tool tip is as follows: height equals the height of the text as specified by the font (i.e., the point of the font) plus 4, and length equals length of the text plus 4.

The tool tips are displayed at predefined locations relative to the controls. In general, tool tips are displayed centered under edit boxes and combo boxes and displayed relative to tool bar buttons at a position where the upper left-hand corner of the tool tip rectangle is 2 pixels to the left of the top left corner of the button and 15 pixels below the hot spot of the mouse **14**. Those skilled in the art will appreciate that tool tips may be displayed at other locations that are adjacent to the tools.

The discussion will now focus on the controls for determining when to display the tool tip. When the mouse cursor **30** is initially positioned to point to a tool bar control (i.e., the first time that the mouse cursor points to a control while it has been in the tool bar **34**), the steps shown in FIG. **3**A are performed. Initially, the window procedure for the tool bar **34** determines that the mouse cursor **30** points to a tool bar control (step **42** in FIG. **3**A). The application program **26** uses a timer to determine whether or not to display a tool tip. This timer may be a system-provided resource that is provided by the operating system **24** or may be a separate component that is provided by the application program **26**. The tip is displayed when the timer counts up to a preset trigger point and the magnitude of the velocity of the mouse cursor is an acceptable range. The timer trigger point is then set to 0.7 seconds (step **43** in FIG. **3**A). Those skilled in the art will appreciate that the choice of 0.7 seconds is not intended to be limiting of the present invention; rather, 0.7 seconds is a value used in the preferred embodiment of the present invention, which appears to empirically produce desirable results. The timer is then reset to zero seconds so that it can begin counting time.

As mentioned above, the magnitude of the velocity of the mouse cursor **30** is also used to control whether a tool tip is displayed. FIG. **3**B shows the steps that are performed to determine the magnitude of velocity of the mouse cursor **30**. Initially, a mouse message is received (step **45**). As discussed above, the mouse message includes the (X, Y) coordinates that specify the most recent position of the

mouse cursor **30**. The coordinates are utilized to determine whether the mouse cursor **30** points to a tool bar control (step **46**). As was discussed above, the system knows the locations of the tool bar controls within the tool bar **34**. This knowledge is used to determine whether the cursor points to a tool bar control. Mouse messages are generated periodically as the mouse cursor position changes. The time interval between mouse messages is reasonably fixed. Hence, the magnitude of the velocity of the mouse cursor **30** may be determined by comparing the (X, Y) coordinates for the most recently received mouse message with the coordinates from the last previous mouse message. The Euclidean distance between these two sets of coordinates may be calculated and, since the time interval is known, the magnitudes of velocity can then be calculated. However, since the time interval is fixed, there is no need to calculate the magnitude of velocity in each instance; rather, the measure of the distance traveled quantifies the magnitude of the velocity of the mouse cursor **30**. Accordingly, the preferred embodiment of the present invention merely measures the distance in pixels since the last mouse message as the velocity metric (step **47**).

Those skilled in the art will appreciate that in alternative embodiments, the present invention may use the magnitude of the velocity as the measure that must exceed a predetermined empirically derived threshold. Alternatively, the vector value of the velocity may be utilized and compared to a vector threshold to determine whether a tool tip should be displayed or not. Moreover, those skilled in the art will appreciate that mouse cursor velocity may be used alone to determine whether a tool tip is displaced or, as employed in the preferred embodiment of the present invention, may be used in conjunction with time metrics to determine whether to display a tool tip or not. In addition, it should be realized that the controls described herein may be applied more broadly to any events that are triggered by measuring the time and velocity variables or velocity variable alone of a mouse cursor within a region of a user interface.

Whenever the mouse cursor **30** is positioned to point to a control within the tool bar **34** of the application program **26**, the steps shown in FIG. **4** are performed. Initially, the timer is initiated to run (step **48**). When the timer expires (step **50**), a determination is made whether the mouse cursor **30** is still pointing at the same control on the tool bar **34** (step **51** in FIG. **4**) and whether the measured velocity metric has remained below an empirically derived threshold value during the time period (step **52**). If both of these conditions are met, a tool tip is displayed as discussed above (step **54** in FIG. **4**). In an alternative embodiment, the velocity metric is only measured as the timer expires rather than during the entire time interval. In addition to the tool tip being displayed, the timer trigger point is set to 0.1 seconds (step **56**). The setting of the timer trigger point to a shorter duration (e.g., 0.1 seconds) facilitates browsing, so that the tool tips will be more quickly displayed when the user moves to an adjacent button or other control on the tool bar **34**. If either of the conditions is not met, a tool tip is not displayed and the timer is reset to 0 seconds (step **53**).

The time duration for which the mouse cursor **30** remains pointing at a tool bar control and the measured magnitude of the velocity of the mouse cursor provide helpful indicators of the intent of the user. Empirical tests indicate that users typically will leave the mouse cursor pointing at a control for a time period greater than 0.7 seconds if they wish to use the control. Similarly, users tend to move the mouse cursor slowly over a tool bar control when they wish to utilize the tool bar control. In contrast, when users do not wish to use a tool bar control and are merely passing over a tool bar control, the users move the mouse cursor with a sufficient magnitude of velocity to indicate their intent.

FIG. **5** is a flow chart illustrating the steps performed when a mouse cursor is positioned to no longer point at a tool bar control (step **58**). As soon as the cursor no longer points at the tool bar control, the tool tip is no longer displayed (step **60**).

When the mouse cursor is positioned so as to no longer point within the tool bar, the steps shown in FIG. **6** are performed. In particular, when the cursor leaves the tool bar (step **62**), the timer is cleared (step **64**). The clearing of the timer allows the processing to be reinitiated when the cursor again returns to point to a location within the tool bar.

While the present invention has been described with reference to a preferred embodiment thereof, those skilled in the art will appreciate that various changes in form and detail may be made without departing from the intended scope of the present invention as defined in the appended claims. For example, the timing control described above relative to the preferred embodiment of the present invention may also be applied to graphical objects other than tools on a tool bar. Moreover, the timing parameters utilized by the preferred embodiment of the present invention are intended to merely illustrative. Other timing parameters may be utilized as well. Still further, pointing devices other than the mouse may be used to position the cursor.

What is claimed is:

1. A method of displaying information about a graphical object displayed on a video display, the method comprising:

determining that a cursor is positioned to point at the graphical object on the video display;

setting a first trigger point representing a predetermined period of time, the predetermined period of time being greater than an interval between two consecutive mouse location events;

determining expiration of the predetermined period of time represented by the first trigger point responsive to the operation of determining that the cursor is positioned to point at the graphical object; and

displaying the information adjacent to the graphical object on the video display, responsive to the operation of determining expiration of the predetermined period of time.

2. The method of claim **1** wherein the operation of determining that a cursor is positioned to point at the graphical object comprises:

receiving a message notification indicating that the cursor is positioned to point at the graphical object on the video display.

3. The method of claim **1** wherein the operation of setting the first trigger point comprises:

initiating a timer to count for the predetermined period of time represented by the first trigger point.

4. The method of claim **3** wherein the operation of determining expiration of the predetermined period of time comprises:

determining that the timer has counted for the predetermined period of time.

5. The method of claim **4** further comprising:

clearing the timer if the cursor is no longer positioned to point at the graphical object before the timer expires.

6. The method of claim **4** further comprising:

clearing the timer if the cursor is moved before the timer expires.

7. The method of claim 1 wherein the displaying operation comprises;

determining a position of the cursor after the predetermined period of time expires;

determining whether the position of the cursor is over the graphical object on the video display, responsive to the operation of determining the position of the cursor, and

displaying on the video display the information about the graphical object at a predefined location relative to the graphical object.

8. The method of claim 1 further comprising:

terminating display of the information if the cursor no longer points at the graphical object on the video display, after the displaying operation.

9. The method of claim 1 further comprising:

determining movement of the cursor to point at a different graphical object displayed on the video display;

setting a second trigger point representing a shorter period of time than the predetermined period of time, responsive to the operation of displaying the information adjacent to the different graphical object on the video display;

detecting expiration of the shorter period of time representing the second trigger point, responsive to the operation of determining movement of the cursor to point at the different graphical object; and

displaying the information adjacent to the different graphical object on the video display, responsive to the operation of detecting expiration of the shorter period of time.

10. The method of claim 1 further comprising:

determining the information to be displayed about the graphical object, responsive to the operation of determining expiration of the predetermined period of time.

11. A method of displaying information about a control displayed on a video display, the method comprising:

determining that a cursor is positioned to point at the control on the video display;

waiting a predetermined period of time, the predetermined period of time being greater than an interval between two consecutive mouse location events;

receiving notification that the predetermined period of time has expired; and

displaying the information adjacent to the control on the video display, if the cursor is still positioned to point at the control after expiration of the predetermined period of time.

12. The method of claim 11 wherein the operation of displaying the information comprises:

displaying text about the control adjacent to the control on the video display.

13. The method of claim 11 wherein the control includes a button control.

14. The method of claim 11 wherein the control includes a list box control.

15. The method of claim 11 wherein the operation of displaying the information comprises:

determining a velocity of the cursor over the control; and

displaying the information adjacent to the control on the video display, if the cursor is still positioned to point at the control after expiration of the predetermined period of time and the velocity remains below a predetermined threshold.

16. A computer program storage medium readable by a computer system and encoding a computer program for executing a computer process for displaying information about a graphical object displayed on a video display, the computer process comprising:

determining that a cursor is positioned to point at the graphical object on the video display;

setting a first trigger point representing a predetermined period of time, the predetermined period of time being greater than an interval between two consecutive mouse location events;

determining expiration of the predetermined period of time represented by the first trigger point, responsive to the operation of determining that the cursor is positioned to point at the graphical object; and

displaying the information adjacent to the graphical object on the video display, responsive to the operation of determining expiration of the predetermined period of time.

17. The computer program storage medium of claim 16 wherein the operation of determining that a cursor is positioned to point at the graphical object comprises:

receiving a message notification indicating that the cursor is positioned to point at the graphical object on the video display.

18. The computer program storage medium of claim 16 wherein the operation of setting the first trigger point comprises:

initiating a timer to count for the predetermined period of time represented by the first trigger point.

19. The computer program storage medium of claim 18 wherein the operation of determining expiration of the predetermined period of time comprises:

determining that the timer has counted for the predetermined period of time.

20. The computer program storage medium of claim 19 wherein the computer process further comprises:

clearing the timer if the cursor is no longer positioned to point at the graphical object before the timer expires.

21. The computer program storage medium of claim 19 wherein the computer process further comprises:

clearing the timer if the cursor is moved before the timer expires.

22. The computer program storage medium of claim 16 wherein the displaying operation comprises;

determining a position of the cursor after the predetermined period of time expires;

determining whether the position of the cursor is over the graphical object on the video display, responsive to the operation of determining the position of the cursor; and

displaying on the video display the information about the graphical object at a predefined location relative to the graphical object.

23. The computer program storage medium of claim 16 wherein the computer process further comprises:

terminating display of the information if the cursor no longer points at the graphical object on the video display, after the displaying operation.

24. The computer program storage medium of claim 16 wherein the computer process further comprises:

determining movement of the cursor to point at a different graphical object displayed on the video display;

setting a second trigger point representing a shorter period of time than the predetermined period of time, responsive to the operation of displaying the information adjacent to the different graphical object on the video display;

determining expiration of the shorter period of time representing the second trigger point, responsive to the operation of determining movement of the cursor to point at the different graphical object; and

displaying the information adjacent to the different graphical object on the video display, responsive to the operation of determining expiration of the shorter period of time.

25. The computer program storage medium of claim **16** wherein the computer process further comprises:

determining the information to be displayed about the graphical object, responsive to the operation of detecting expiration of the predetermined period of time.

26. A computer program storage medium readable by a computer system and encoding a computer program for executing a computer process for displaying information about a graphical object displayed on a video display, the computer process comprising:

detecting a cursor positioned to point at the control on the video display;

waiting a predetermined period of time, the predetermined period of time being greater than an interval between two consecutive mouse location events;

receiving notification that the predetermined period of time has expired; and

displaying the information adjacent to the control on the video display, if the cursor is still positioned to point at the control after expiration of the predetermined period of time.

27. The computer program storage medium of claim **26** wherein the operation of displaying the information comprises:

displaying text about the control adjacent to the control on the video display.

28. The computer program storage medium of claim **26** wherein the control includes a button control.

29. The computer program storage medium of claim **26** wherein the control includes a list box control.

30. The computer program storage medium of claim **26** wherein the operation of displaying the information comprises:

detecting a velocity of the cursor over the control; and

displaying the information adjacent to the control on the video display, if the cursor is still positioned to point at the control after expiration of the predetermined period of time and the velocity remains below a predetermined threshold.

* * * * *

Exhibit G

US006281879B1

(12) **United States Patent**
Graham

(10) **Patent No.: US 6,281,879 B1**
(45) **Date of Patent: Aug. 28, 2001**

(54) **TIMING AND VELOCITY CONTROL FOR DISPLAYING GRAPHICAL INFORMATION**

(75) Inventor: **Christopher E. Graham**, Redmond, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/873,855**

(22) Filed: **Jun. 12, 1997**

Related U.S. Application Data

(63) Continuation of application No. 08/709,529, filed on Sep. 6, 1996, now abandoned, which is a continuation of application No. 08/260,558, filed on Jun. 16, 1994, now abandoned.

(51) **Int. Cl.**[7] **G09G 5/08**
(52) **U.S. Cl.** **345/157**; 345/159
(58) **Field of Search** 345/157, 159, 345/145, 146, 119, 120, 123, 348, 349; 395/155, 156, 157, 159

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,789,962		12/1988	Berry et al.	364/900
4,984,152		1/1991	Muller	364/200
5,140,678	*	8/1992	Torres .	
5,155,806		10/1992	Hoeber et al.	395/157
5,157,768		10/1992	Hoeber et al.	395/157
5,169,342		12/1992	Steele et al.	434/112
5,196,838		3/1993	Meier et al.	340/724
5,287,448	*	2/1994	Nicol et al. .	
5,299,307	*	3/1994	Young	345/157
5,373,309	*	12/1994	Totsuka et al.	345/145
5,546,521	*	8/1996	Martinez .	

OTHER PUBLICATIONS

Macintosh Reference, Apple Computer Inc, pp. 30–31, 1991.*

Quick Result, Microsoft Word, Version 6.0, 1993, pp 39–40 and 154–155.*

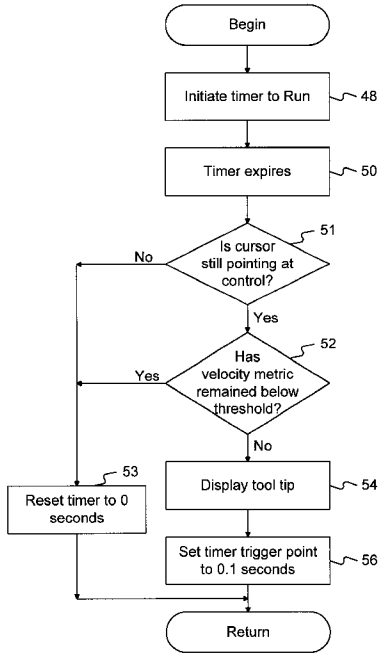
* cited by examiner

Primary Examiner—Chanh Nguyen
(74) *Attorney, Agent, or Firm*—Merchant & Gould, P.C.

(57) **ABSTRACT**

Time and velocity metrics are used to control when information about a graphical object to which a cursor points is displayed on a video display. The time metric is used to ensure that a non-negligible amount of time passes between the time at which the cursor initially points to the graphical object and the time at which the information about the graphical object is displayed on the video display. The time delay helps to eliminate such information being displayed inadvertently when the user quickly passes the cursor over graphical objects in the video display. In addition, the timing control facilitates the shortening of the delay when it appears that the user wishes to browse amongst several related graphical objects that are shown in the video display. For example, when it appears that the user wishes to browse tools on the tool bar, the delay is shortened. The velocity metric is used to determine the likelihood that the user intended to point to the graphical object and serves to minimize instances where undesired information about the graphical object is displayed.

9 Claims, 7 Drawing Sheets



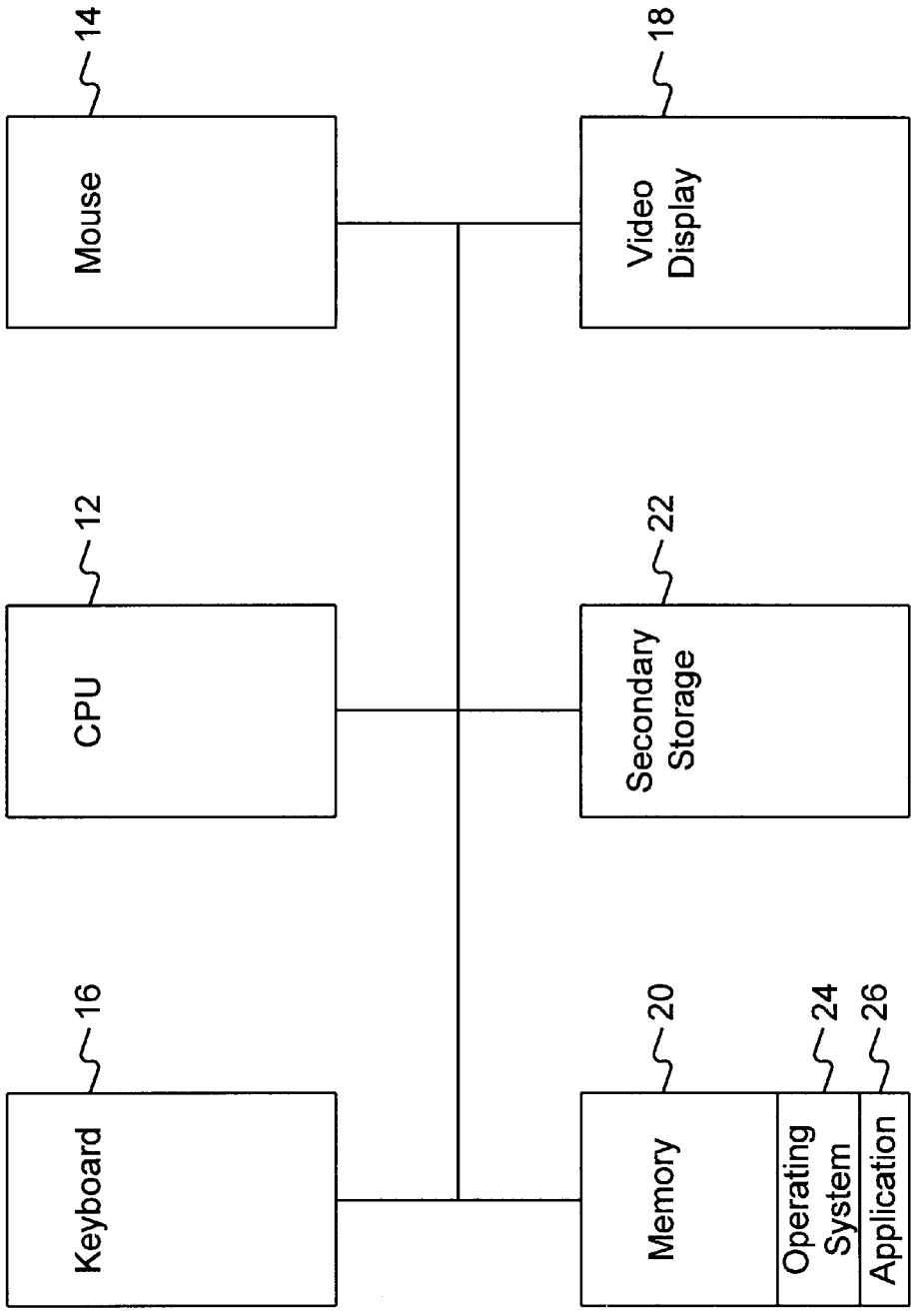

14
Mouse
18
Video Display
12
CPU
22
Secondary Storage
16
Keyboard
20
Memory
24
Operating System
26
Application

FIG. 1

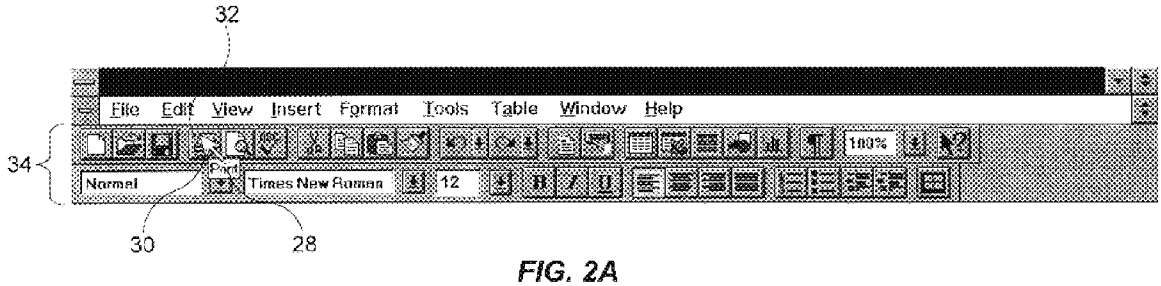

32
File Edit View Insert Format Tools Table Window Help
34
Normal
Times New Roman
12
100%
30
28

FIG. 2A

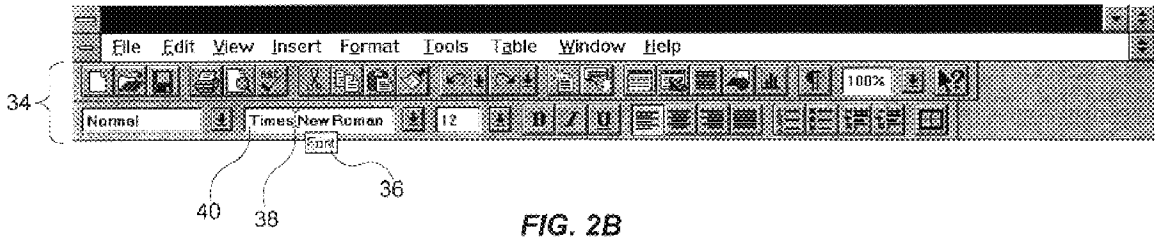

File Edit View Insert Format Tools Table Window Help
34
Normal
Times New Roman
12
100%
40
38
36

FIG. 2B

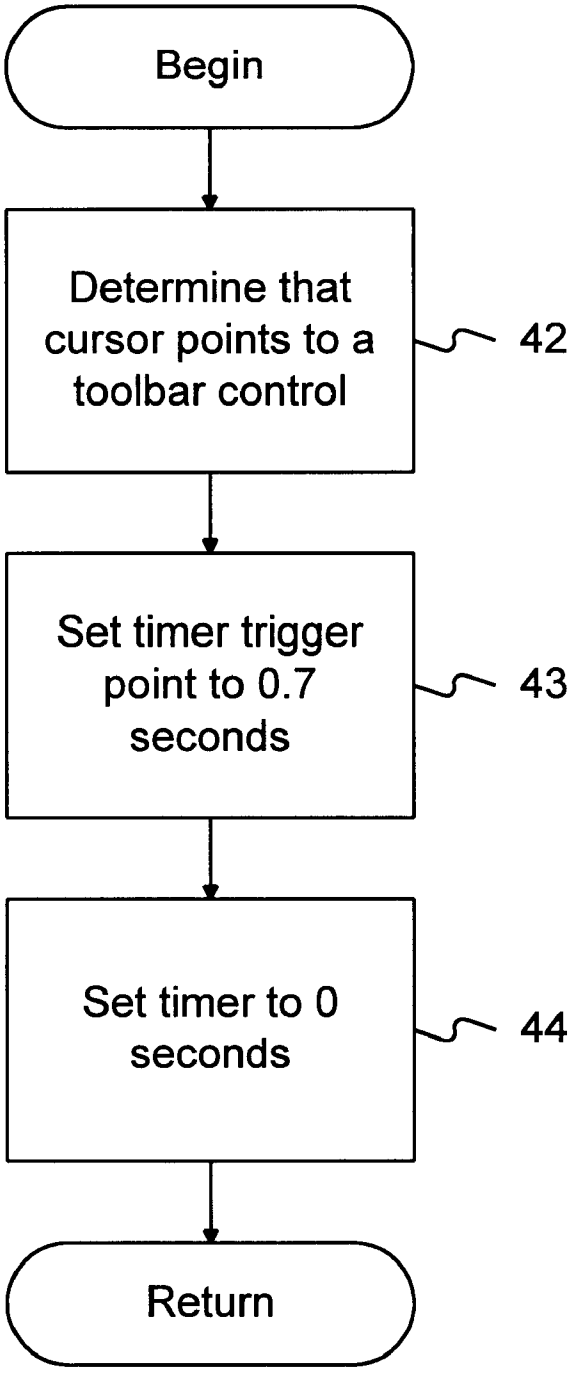

Begin
Determine that cursor points to a toolbar control
42
Set timer trigger point to 0.7 seconds
43
Set timer to 0 seconds
44
Return

FIG. 3A

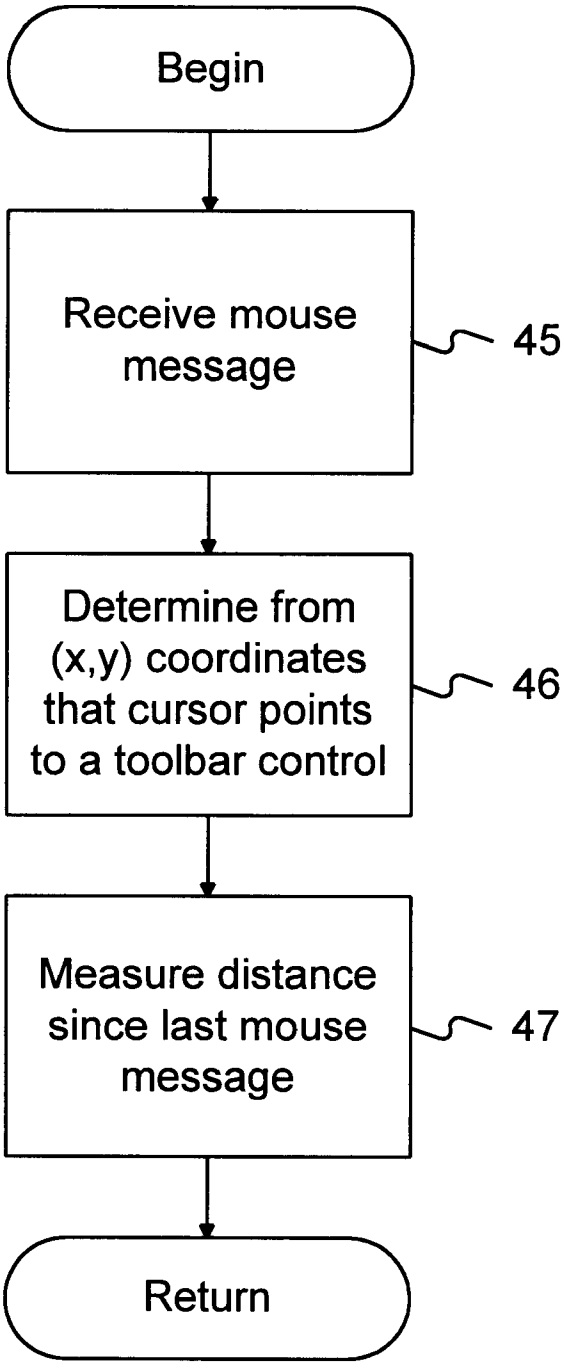

Begin
Receive mouse message
45
Determine from (x,y) coordinates that cursor points to a toolbar control
46
Measure distance since last mouse message
47
Return

FIG. 3B

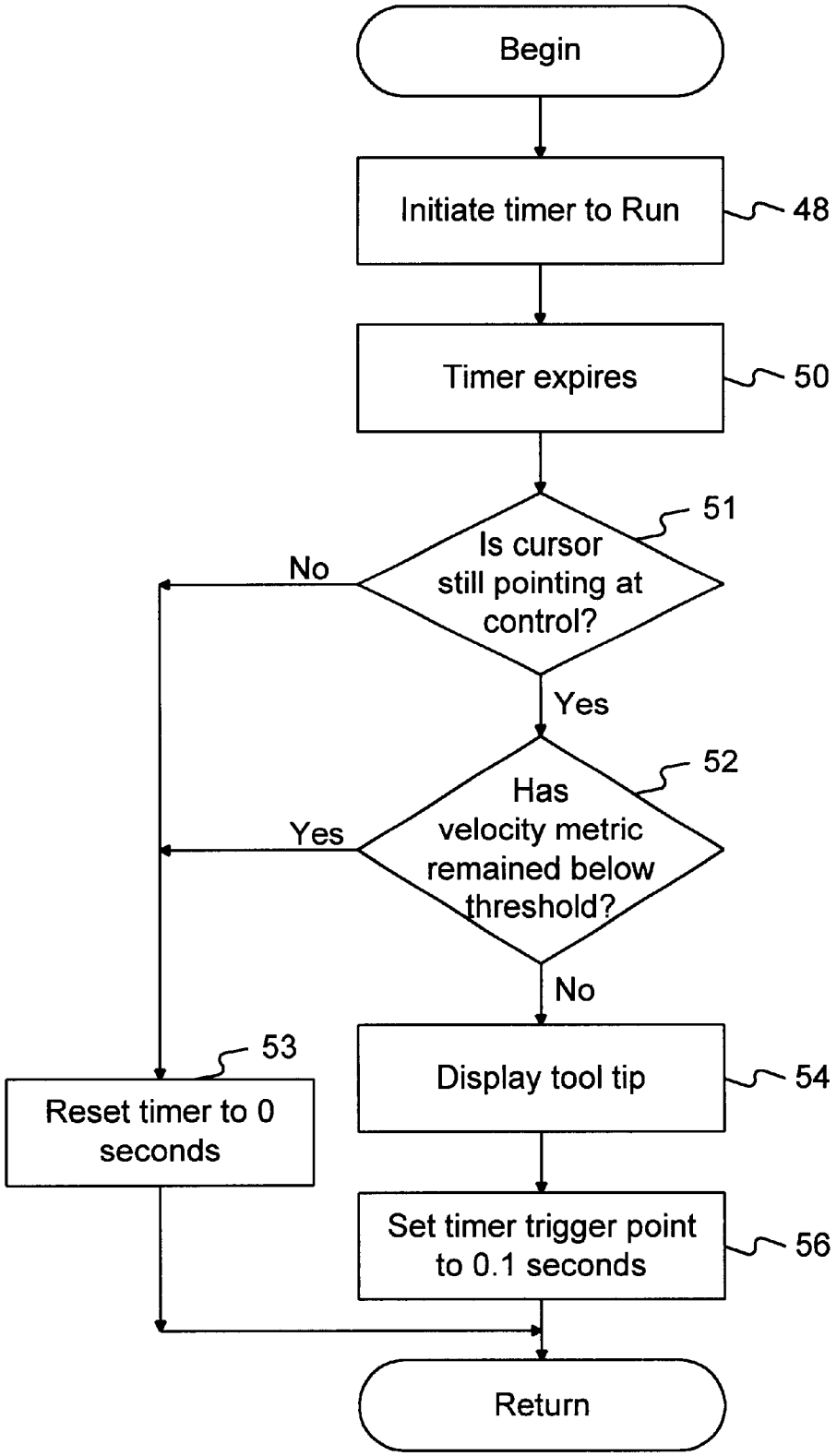

Begin
Initiate timer to Run
48
Timer expires
50
51
Is cursor still pointing at control?
No
Yes
52
Has velocity metric remained below threshold?
Yes
No
53
Reset timer to 0 seconds
Display tool tip
54
Set timer trigger point to 0.1 seconds
56
Return

FIG. 4

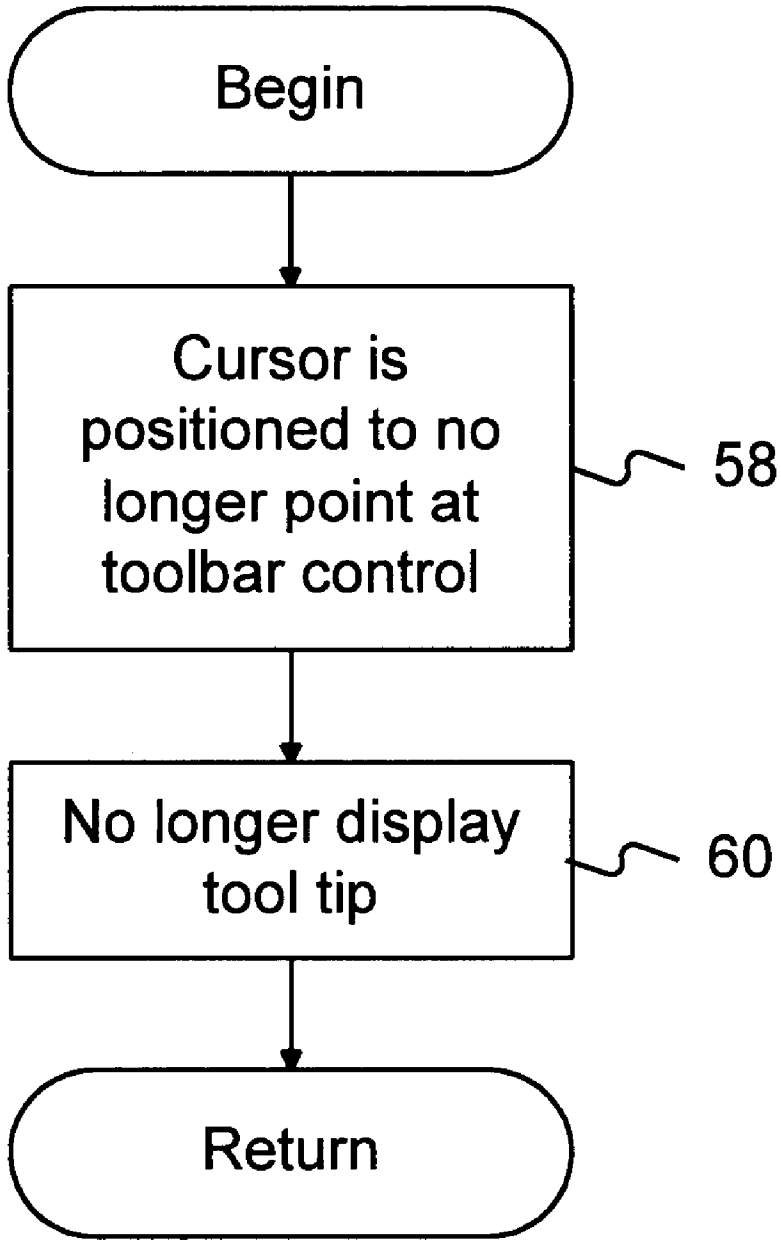

Begin
Cursor is positioned to no longer point at toolbar control
58
No longer display tool tip
60
Return

FIG. 5

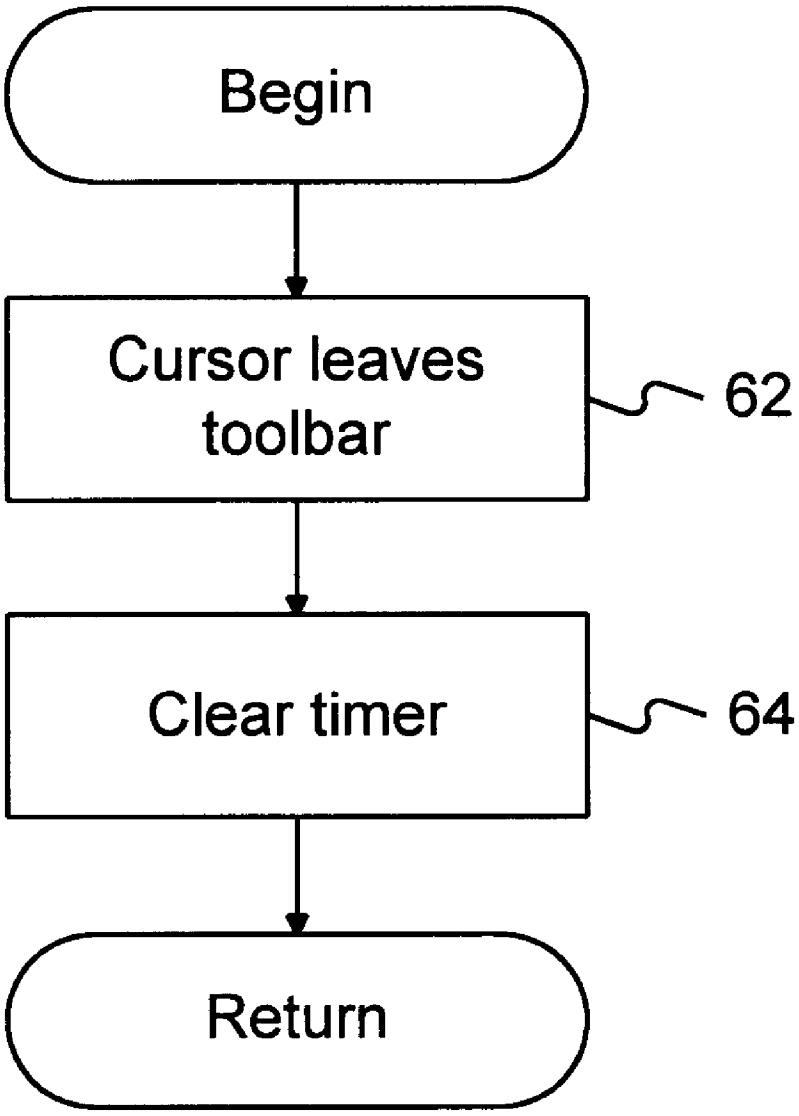


FIG. 6

TIMING AND VELOCITY CONTROL FOR DISPLAYING GRAPHICAL INFORMATION

CROSS-REFERENCE TO RELATED APPLICATION

This application is a continuation of U.S. patent application Ser. No. 08/709,529, filed Sep. 6, 1996, now abandoned, which is a File Wrapper Continuation of U.S. patent application Ser. No. 08/260,558, filed Jun. 16, 1994, now abandoned.

TECHNICAL FIELD

The present invention relates generally to data processing systems and, more particularly, to the displaying of graphical information in data processing systems.

BACKGROUND OF THE INVENTION

Many conventional application programs utilize tool bars. Tool bars provide the user with a number of tools that assist the user in performing tasks.

Typically, a separate control is provided for each tool on the tool bar. The control may be a pushbutton or another graphical object that allows the user to invoke the desired tool. Often times the controls on the tool bar have icons on their faces that indicate the nature of the tool. Unfortunately, as the typical number of controls on the tool bar has grown for applications, it has become more and more difficult for the user to discern the nature of the tool solely from the icons shown as part of the tool bar. As such, many users have difficulty using the tools on the tool bar.

SUMMARY OF THE INVENTION

The limitations of the prior art are overcome by the present invention. In accordance with a first aspect of the present invention, a method is practiced in a data processing system having a video display for displaying a cursor that points to positions on the video display. The data processing system also includes an input device for manipulating the cursor. In accordance with this method, it is first determined that the cursor points to a position within a region on the video display. A velocity metric of the cursor is measured. Where the velocity metric does not exceed a predetermined threshold value, an event is triggered. On the other hand, where the velocity metric exceeds the predetermined threshold value, the event is inhibited.

In accordance with a second aspect of the present invention, it is determined that a cursor points to a position within a region on the video display. A time period metric that specifies how long the cursor has remained pointing within the region is measured. A velocity metric of the cursor within the region is also measured. Based upon these metrics, a determination is made whether to trigger an event.

In accordance with an additional aspect of the present invention, a method is practiced in a data processing system having a video display for displaying a cursor that points to positions in the video display and an input device for manipulating the cursor. In accordance with this method, a graphical object is displayed on the video display. The user uses the input device, and in response, the data processing system positions the cursor to point at the graphical object. A predetermined period of time, such as a time greater than 0.4 seconds, is allowed to pass and then a determination is made whether the cursor still points at the graphical object. If it is determined that the cursor still points at the graphical object, information about the graphical object is displayed adjacent to the graphical object on the video display.

In accordance with another aspect of the present invention, a method is practiced wherein a tool bar having tools is displayed on the video display. When the user uses the input device, the cursor is position to point at a selected one of the tools on the tool bar. The system waits a predetermined non-negligible amount of time. The system also measures a velocity metric of the cursor within the first graphical object. If the cursor still points at the selected tool after waiting the predetermined non-negligible amount of time and the velocity metric has remained below a predetermined threshold during the predetermined non-negligible amount of time, information about the selected tool is displayed in the video display. The position is adjacent to the selected tool.

In accordance with a further aspect of the present invention, a method is practiced in a computer system having a video display for displaying a cursor that points to positions on the video display and an input device for moving the cursor on the video display. In this method, a first graphical object is displayed on the video display. In response to the user using the input device, the cursor is positioned to point at the first graphical object. The system waits a non-negligible predetermined amount of time. A determination is made whether the cursor still points at the graphical object after the non-negligible predetermined amount of time has passed. Where the cursor still points at the first graphical object, a number of steps are performed. These steps include displaying information about the first graphical object adjacent to the first graphical object of the video display. The non-negligible predetermined amount of time is then reset to a substantially shorter amount of time. A second graphical object is displayed on the video display and, in response to the user using the input device, the cursor is positioned to point at the second graphical object on the video display. The system waits the substantially shorter amount of time. Where the cursor is still pointing at the second graphical object after waiting the substantially shorter period of time, information about the second graphical object is displayed adjacent to the second graphical object on the video display.

In accordance with a still further aspect of the present invention, a data processing system includes a video display for displaying video data. The video display displays a first graphical object and a cursor that points to the first graphical object. An input device is included in this part of the data processing system for moving the cursor on the video display. A message generator is provided for displaying information about the first graphical object. The information is displayed adjacent to the first graphical object on the video display when the cursor remains pointing at the first graphical object for a predetermined non-negligible amount of time. The message generator includes a comparator and a message source. The comparator determines whether the cursors remain pointing at the first graphical object for the predetermined non-negligible amount of time. The message source provides and displays information about the first graphical object adjacent to the first graphical object on the video display when the comparator determines that the cursor has remained pointing at the first graphical object for the specified amount of time.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a data processing system that is suitable for practicing a preferred embodiment of the present invention.

FIG. **2**A is a diagram illustrating a tool bar and a tool tip that is provided for a print button in accordance with the preferred embodiment of the present invention.

FIG. **2**B is a diagram illustrating a tool bar and a tool tip that is provided for a font list box in accordance with the preferred embodiment of the present invention.

FIG. **3**A is a flow chart illustrating the steps performed to initially set a timer when a cursor is positioned over a control on the tool bar in accordance with the preferred embodiment of the present invention.

FIG. **3**B is a flow chart illustrating the steps performed to determine the magnitude of the velocity of the cursor when the cursor is positioned over a control on the tool bar in accordance with the preferred embodiment of the present invention.

FIG. **4** is a flow chart illustrating the steps performed to determine whether a tool tip is to be displayed in the preferred embodiment of the present invention.

FIG. **5** is a flow chart illustrating the steps performed when a cursor no longer points within the tool bar in the preferred embodiment of the present invention.

FIG. **6** is a flow chart illustrating the steps that are performed relative to the timer when the cursor leaves the tool bar in the preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The preferred embodiment of the present invention displays a tool tip when a mouse cursor points to a tool or a tool bar for a sufficient amount of time and the magnitude of the velocity of the mouse cursor remains below a predetermined threshold. A tool tip is a brief textual message, such as a name of a tool, that identifies the nature of the tool. The preferred embodiment of the present invention provides a delay between when the mouse cursor is initially positioned to point at the tool control on the tool bar and when the tool tip is displayed. This delay prevents the user from receiving undesired tool tips when the user inadvertently passes the mouse cursor over a control on the tool bar. The delay is sufficiently long (i.e., it is non-negligible) to allow the user to move the mouse cursor if he does not want to receive a tool tip. The delay is shortened when an initial tool tip is displayed so as to allow the user to quickly browse the tool controls that are available on the tool bar. The magnitude of the velocity of the mouse cursor is measured to determine whether the user likely intends to point at the tool to receive a tool tip or whether the user, instead, is merely passing over the tool while moving to another destination.

Although the preferred embodiment of the present invention concerns controlling when tool tips for tools on a tool bar are displayed, those skilled in the art will appreciate that the present invention may generally be applied to other regions of a video display (such as other types of graphical objects) for which graphical information is to be provided. It should be appreciated that the present invention may be applied to both visible and invisible graphical objects.

FIG. **1** is a block diagram of a data processing system **10** that is suitable for practicing the preferred embodiment of the present invention. The data processing system **10** includes at least one central processing unit (CPU) **12**. The CPU **12** is connected to a number of peripheral devices, including a mouse **14**, a keyboard **16** and a video display **18**. The CPU **12** is also connected to a memory **20** and a secondary storage device **22**, such as a hard disk drive. The memory **20** holds a copy of an operating system **24**, such as the Microsoft Windows, version 3.1, operating system sold by Microsoft Corporation of Redmond, Wash. The memory **20** also holds a copy of an application program **26**. The

implementation of the preferred embodiment of the present invention will be described below with reference to use of tool tips within the application program **26**. Nevertheless, it should be appreciated that the tool tips may alternatively be implemented in the operating system **24** or as a system resource.

FIG. **2**A shows the tool bar **34** that is generated and displayed on the video display **18** when the application program **26** is run on the CPU **12**. The tool bar **34** includes a number of controls, such as buttons and list boxes, that enable a user to access the tools of the tool bar. When the user positions a mouse cursor **30** over one of the buttons and clicks the mouse (i.e., quickly depresses and releases a predefined one of the mouse buttons), the tool associated with the button is invoked. Similarly, by positioning the mouse cursor **30** over one of the buttons of the list boxes, a drop-down list appears, and the user may select one of the options on the drop-down list using the mouse **14**.

The tool bar **34** is created as a window by the application program **26**. The operating system **24** facilitates the definition of such windows (as is provided in the Microsoft Windows, version 3.1, operating system). FIG. **2**A shows the mouse cursor **30** pointing to a print button **32** on the tool bar **34**. When a user positions the mouse cursor **30** over the print button **32** and clicks the mouse button, the document currently displayed in the window of the application program is printed.

The user interface provided for the application program **24** is logically divisible into a number of windows. One of these windows is the tool bar **34**. In general, each window of the user interface has a separate window procedure associated with it. The operating system **24** maintains a message queue for each program that generates windows. Accordingly, the application program **26** has its own message queue. Since the application program **26** may generate multiple windows, the message queue may hold messages for multiple windows. When an event occurs, the event is translated into a message that is put into the message queue for the application program **26**. The application program **26** retrieves and delivers the message to the proper windows by executing a block code known as the "message loop." The window procedure that receives the message then processes the message.

Movements of the mouse **14** are reflected in messages that are placed into the message queue of the application program **26**. In particular, when a user positions the mouse cursor **30** with the mouse **14** over a window or clicks the mouse by depressing one of the mouse buttons within a window, the procedure for the window receives a mouse message. The operating system **24** provides a number of predefined mouse messages. The mouse messages specify the status of mouse buttons and the position of the mouse cursor **30** within the window. The position of the mouse cursor **30** within the window is specified in (X, Y) coordinates relative to the upper left-hand corner of the window. Thus, when the mouse cursor **30** moves within the tool bar **34**, the position of the mouse cursor **30** within the tool bar is reflected and a mouse message that specifies (X, Y) coordinates of the mouse cursor relative to the upper left-hand corner of the tool bar. The window procedure receives the mouse message and utilizes the information contained in the message to respond to the mouse **14** activities.

As mentioned above, the application program **26** specifies the window that constitutes the tool bar **34**. The application program **26** paints each of the controls, including print button **32**, at known locations within the window of the tool

bar **34**. When the mouse cursor **30** is positioned within the tool bar, mouse messages specify the position of the mouse cursor within the tool bar **34**. The mouse messages are sent to the window procedure that is responsible for the tool bar window. The window procedure for the tool bar **34** compares the coordinates specified by the mouse message with the known location of the controls within the tool bar. Accordingly, the window procedure for the tool bar **34** can determine whether the mouse cursor **30** is pointing at any of the controls. When it is determined that the mouse cursor **30** is pointing at one of the controls of the tool bar **34**, a tool tip **28** may be displayed if the mouse cursor **30** has remained pointing at the control for a sufficient period of time and the magnitude of the velocity of the mouse cursor is below a predetermined threshold. Hence, in the example shown in FIG. **2**A, the message "Print" is displayed as a tool tip **28**, given that the mouse cursor **30** is pointing to the print button **32**.

Tool tips are provided not only for buttons on the tool bar **34** but are also provided for other types of controls. For example, as shown in FIG. **2**B, a mouse cursor **38** points to a portion of a list box **40** that concerns the font which the user wishes to utilize. Accordingly, a tool tip **36** is displayed that includes the text "Font". It is worth noting that the mouse cursor **38** changes from an arrow to a cross bar, since the cursor points to a portion of a list box that contains text rather than a button as in FIG. **2**A.

Tool tips are displayed using text output commands that are provided by the operating system **24**. Specifically, the ExtTextOut ( ) function that is provided by the Microsoft Windows, version 3.1, operating system is used in the preferred embodiment. The format of this on is as follows:

BOOL ExtTextOut(hdc, nXStart, nYStart, fuOptions, lprc, lpszString, cbString, lpDx)

HDC hdc;	/* handle of device context	*/
int nXStart;	/* x-coordinate of starting position	*/
int nYStart;	/* y-coordinate of starting position	*/
UINT fuOptions;	/* rectangle type	*/
const RECT FAR* lprc;	/* address of structure with rectangle	*/
LPCSTR lpszString;	/* address of string	*/
UINT cbString;	/* number of bytes in string	*/
int FAR* lpDx;	/* spacing between character cells	*/

The hdc parameter of this function specifies a handle (i.e., a numerical identifier) for a device context. In this case, the device context specifies attributes that determine how the operating system interacts with the video display **18**. The nXStart parameter specifies the logical X coordinate at which the string of the tool tip message begins. Similarly, the nYStart parameter specifies the logical Y coordinate at which the string begins. The fuOptions parameter specifies the type of rectangle for the tool tip. The operating system **24** provides predefined data structures that specify rectangle types. In this case, the rectangle type is defined as a clipped rectangle. The lprc parameter is a pointer to a structure that holds a rectangle and the lpszstring is a pointer to a structure that holds the textual string to be displayed in the tool tip. The cbstring parameter specifies the number of bytes in the string and the lpDx parameter specifies spacing between character cells.

When the window procedure for the tool bar **34** receives a mouse message that indicates that the mouse cursor **30** (see FIG. **2**A) is positioned over one of the controls of the tool bar **34**, a sufficient time has elapsed and the measured magnitude of the velocity of the mouse cursor **30** is below a predetermined threshold, the procedure determines the string that is to be displayed in the tool tip for the control to which the mouse cursor points. The address of this string is passed as the lpszstring parameter to the ExtTextOut ( ) function. This function then proceeds to draw the tool tip.

As the rectangle for the tool tip is a clipped rectangle, the background color may be specified. In the preferred embodiment of the present invention, the background color is yellow, as specified in red/green/blue (RGB) coordinates as (**255**, **255**, **128**). The size of the rectangle used for the tool tip is as follows: height equals the height of the text as specified by the font (i.e., the point of the font) plus 4, and length equals length of the text plus 4.

The tool tips are displayed at predefined locations relative to the controls. In general, tool tips are displayed centered under edit boxes and combo boxes and displayed relative to tool bar buttons at a position where the upper left-hand corner of the tool tip rectangle is 2 pixels to the left of the top left corner of the button and 15 pixels below the hot spot of the mouse **14**. Those skilled in the art will appreciate that tool tips may be displayed at other locations that are adjacent to the tools.

The discussion will now focus on the controls for determining when to display the tool tip. When the mouse cursor **30** is initially positioned to point to a tool bar control (i.e., the first time that the mouse cursor points to a control while it has been in the tool bar **34**), the steps shown in FIG. **3**A are performed. Initially, the window procedure for the tool bar **34** determines that the mouse cursor **30** points to a tool bar control (step **42** in FIG. **3**A). The application program **26** uses a timer to determine whether or not to display a tool tip. This timer may be a system-provided resource that is provided by the operating system **24** or may be a separate component that is provided by the application program **26**. The tip is displayed when the timer counts up to a preset trigger point and the magnitude of the velocity of the mouse cursor is an acceptable range. The timer trigger point is then set to 0.7 seconds (step **43** in FIG. **3**A). Those skilled in the art will appreciate that the choice of 0.7 seconds is not intended to be limiting of the present invention; rather, 0.7 seconds is a value used in the preferred embodiment of the present invention, which appears to empirically produce desirable results The timer is then reset to zero seconds so that it can begin counting time.

As mentioned above, the magnitude of the velocity of the mouse cursor **30** is also used to control whether a tool tip is displayed. FIG. **3**B shows the steps that are performed to determine the magnitude of velocity of the mouse cursor **30**. Initially, a mouse message is received (step **45**). As discussed above, the mouse message includes the (X, Y) coordinates that specify the most recent position of the mouse cursor **30**. The coordinates are utilized to determine whether the mouse cursor **30** points to a tool bar control (step **46**). As was discussed above, the system knows the locations of the tool bar controls within the tool bar **34**. This knowledge is used to determine whether the cursor points to a tool bar control. Mouse messages are generated periodically as the mouse cursor position changes. The time interval between mouse messages is reasonably fixed. Hence, the magnitude of the velocity of the mouse cursor **30** may be determined by comparing the (X, Y) coordinates for the most recently received mouse message with the coordinates from the last previous mouse message. The Euclidean distance between these two sets of coordinates may be calculated and, since the time interval is known, the magnitudes of velocity can then be calculated. However, since the time interval is fixed, there is no need to calculate the magnitude

of velocity in each instance; rather, the measure of the distance traveled quantifies the magnitude of the velocity of the mouse cursor **30**. Accordingly, the preferred embodiment of the present invention merely measures the distance in pixels since the last mouse message as the velocity metric (step **47**).

Those skilled in the art will appreciate that in alternative embodiments, the present invention may use the magnitude of the velocity as the measure that must exceed a predetermined empirically derived threshold. Alternatively, the vector value of the velocity may be utilized and compared to a vector threshold to determine whether a tool tip should be displayed or not. Moreover, those skilled in the art will appreciate that mouse cursor velocity may be used alone to determine whether a tool tip is displayed or, as employed in the preferred embodiment of the present invention, may be used in conjunction with time metrics to determine whether to display a tool tip or not. In addition, it should be realized that the controls described herein may be applied more broadly to any events that are triggered by measuring the time and velocity variables or velocity variable alone of a mouse cursor within a region of a user interface.

Whenever the mouse cursor **30** is positioned to point to a control within the tool bar **34** of the application program **26**, the steps shown in FIG. **4** are performed. Initially, the timer is initiated to run (step **48**). When the timer expires (step **50**), a determination is made whether the mouse cursor **30** is still pointing at the same control on the tool bar **34** (step **51** in FIG. **4**) and whether the measured velocity metric has remained below an empirically derived threshold value during the time period (step **52**). If both of these conditions are met, a tool tip is displayed as discussed above (step **54** in FIG. **4**). In an alternative embodiment, the velocity metric is only measured as the timer expires rather than during the entire time interval. In addition to the tool tip being displayed, the timer trigger point is set to 0.1 seconds (step **56**). The setting of the timer trigger point to a shorter duration (e.g., 0.1 seconds) facilitates browsing, so that the tool tips will be more quickly displayed when the user moves to an adjacent button or other control on the tool bar **34**. If either of the conditions is not met, a tool tip is not displayed and the timer is reset to 0 seconds (step **53**).

The time duration for which the mouse cursor **30** remains pointing at a tool bar control and the measured magnitude of the velocity of the mouse cursor provide helpful indicators of the intent of the user. Empirical tests indicate that users typically will leave the mouse cursor pointing at a control for a time period greater than 0.7 seconds if they wish to use the control. Similarly, users tend to move the mouse cursor slowly over a tool bar control when they wish to utilize the tool bar control. In contrast, when users do not wish to use a tool bar control and are merely passing over a tool bar control, the users move the mouse cursor with a sufficient magnitude of velocity to indicate their intent.

FIG. **5** is a flow chart illustrating the steps performed when a mouse cursor is positioned to no longer point at a tool bar control (step **58**). As soon as the cursor no longer points at the tool bar control, the tool tip is no longer displayed (step **60**).

When the mouse cursor is positioned so as to no longer point within the tool bar, the steps shown in FIG. **6** are performed. In particular, when the cursor leaves the tool bar (step **62**), the timer is cleared (step **64**). The clearing of the timer allows the processing to be reinitiated when the cursor again returns to point to a location within the tool bar.

While the present invention has been described with reference to a preferred embodiment thereof, those skilled in the art will appreciate that various changes in form and detail may be made without departing from the intended scope of the present invention as defined in the appended claims. For example, the timing control described above relative to the preferred embodiment of the present invention may also be applied to graphical objects other than tools on a tool bar. Moreover, the timing parameters utilized by the preferred embodiment of the present invention are intended to merely illustrative. Other timing parameters may be utilized as well. Still further, pointing devices other than the mouse may be used to position the cursor.

What is claimed is:

1. In a data processing system having a video display for displaying a cursor that points at positions on the video display and an input device for manipulating the cursor, a method comprising the steps of:

displaying a tool bar having tools on the video display;

in response to the user using the input device, positioning the cursor to point at a selected one of the tools on the tool bar;

waiting a predetermined non-negligible amount of time;

measuring a velocity metric of the cursor within the selected tool during the predetermined non-negligible amount of time; and

if the cursor still points at the selected tool and the velocity metric remains below a predetermined threshold during the predetermined non-negligible amount of time, displaying information about the selected tool on the video display adjacent to the selected tool;

changing the predetermined non-negligible amount of time to a new period of time;

in response to the user using the input device, positioning the cursor to point to a second of the tools on the tool bar;

waiting the new period of time;

measuring a velocity metric of the cursor within the second of the tools during the new period of time; and

when the cursor still points at the second of the tools on the tool bar and the velocity metric within the second of the tools has remained below the predetermined threshold during the new period of time, displaying information about the second tool on the video display adjacent to the selected tool.

2. The method of claim **1** wherein displaying information about the selected tool on the video display comprises the step of displaying text about the selected tool on the video display adjacent to the selected tool.

3. The method of claim wherein each of the tools has a name and the text comprises the name of the selected tool.

4. The method of claim **1** wherein the predetermined non-negligible amount of time is at least 0.4 seconds.

5. The method of claim **1** wherein the new period of time is substantially less than the predetermined non-negligible amount of time.

6. In a computer system having a video display for displaying a cursor that points to positions on the video display and an input device for moving the cursor on the video display, a method comprising the steps of:

displaying a first graphical object on the video display;

in response to the user using the input device, positioning the cursor to point at the first graphical object;

waiting a non-negligible predetermined amount of time;

determining whether the cursor still points at the first graphical object after the non-negligible predetermined amount of time; and

where the cursor still points at the first graphical object after the non-negligible predetermined amount of time, displaying information about the first graphical object adjacent to the first graphical object on the video display;

resetting the non-negligible predetermined amount of time to a substantially shorter amount of time;

in response to the user using the input device, positioning the cursor to point at a second graphical object on the video display;

waiting the substantially shorter amount of time; and

where the cursor is still pointing at the second graphical object after the substantially shorter amount of time, displaying information about the second graphical object adjacent to the second graphical object on the video display.

7. The method of claim **6** wherein the first graphical object is a tool on a tool bar.

8. The method of claim **6** wherein the second graphical object is a tool on a tool bar.

9. The method of claim **6** wherein the non-negligible predetermined amount of time is at least 0.4 seconds.

* * * * *

Exhibit H

US005845077A

United States Patent [19]

Fawcett

[11] **Patent Number: 5,845,077**

[45] **Date of Patent: *Dec. 1, 1998**

[54] **METHOD AND SYSTEM FOR IDENTIFYING AND OBTAINING COMPUTER SOFTWARE FROM A REMOTE COMPUTER**

[75] Inventor: **Philip E. Fawcett**, Duvall, Wash.

[73] Assignee: **Microsoft Corporation**, Redmond, Wash.

[*] Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

[21] Appl. No.: **562,929**

[22] Filed: **Nov. 27, 1995**

[51] **Int. Cl.**[6] **G06F 7/00**

[52] **U.S. Cl.** **395/200.51**; 364/221; 364/221.7; 364/222.81; 364/259; 395/712; 395/200.09

[58] **Field of Search** 395/712, 200.09, 395/200.51, 232; 364/479.01; 380/4

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,796,181	1/1989	Wiedemer	364/406
5,047,928	9/1991	Wiedemer	364/406
5,142,680	8/1992	Ottman et al.	395/712
5,155,484	10/1992	Chambers, IV .	
5,155,680	10/1992	Wiedemer	364/406
5,155,847	10/1992	Kirouac et al.	395/200.09
5,267,171	11/1993	Suzuki et al.	364/479.04
5,337,360	8/1994	Fischer .	
5,367,686	11/1994	Fisher et al. .	
5,388,211	2/1995	Hornbuckle	395/712
5,390,247	2/1995	Fischer .	
5,421,009	5/1995	Platt	395/712
5,473,772	12/1995	Halliwell et al. .	
5,495,411	2/1996	Ananda	395/232
5,528,490	6/1996	Hill .	
5,548,645	8/1996	Ananda	380/4
5,586,304	12/1996	Stupeck, Jr. et al. .	
5,586,322	12/1996	Beck et al.	707/200
5,654,901	8/1997	Boman .	

OTHER PUBLICATIONS

Mori et al., "Superdistribution: The Concept and the Architecture", *The Transaction of the Ieice*, vol. E73, No. 7, pp. 1133–1146 (Jul. 1990).

Williams, "Internet Component Download," *Microsoft Interactive Development*, 49–52, Summer, 1996.

Rozenbilt, "O,A & M Capabilities for Switching Software Management" IEEE Global Telecommunications Conference, 1993, pp. 357–361.

Primary Examiner—Tod R. Swann
Assistant Examiner—Fred F. Tzeng
Attorney, Agent, or Firm—Klarquist Sparkman Campbell Leigh & Whinston, LLP

[57] **ABSTRACT**

Creators of computer software provide the most up-to-date versions of their computer software on an update service. A user who has purchased computer software calls the update service on a periodic basis. The update service automatically inventories the user computer to determine what computer software may be out-of-date, and/or need maintenance updates. If so desired by the user, the update service computer automatically downloads and installs computer software to the user computer. By making periodic calls to the update service, the user always has the most up-to-date computer software immediately available. The update service may also alert the user to new products (i.e. including new help files, etc.), and new and enhanced versions of existing products which can be purchased electronically by a user from the update service.

24 Claims, 5 Drawing Sheets

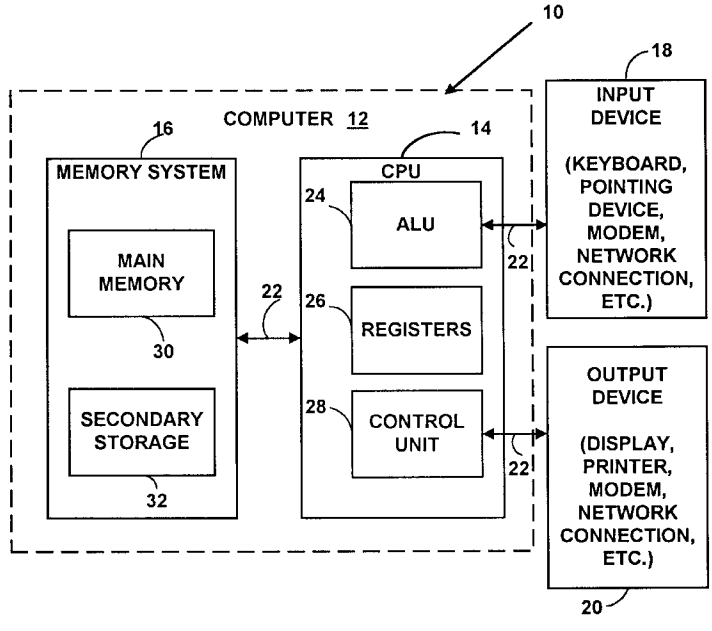


FIG. 1

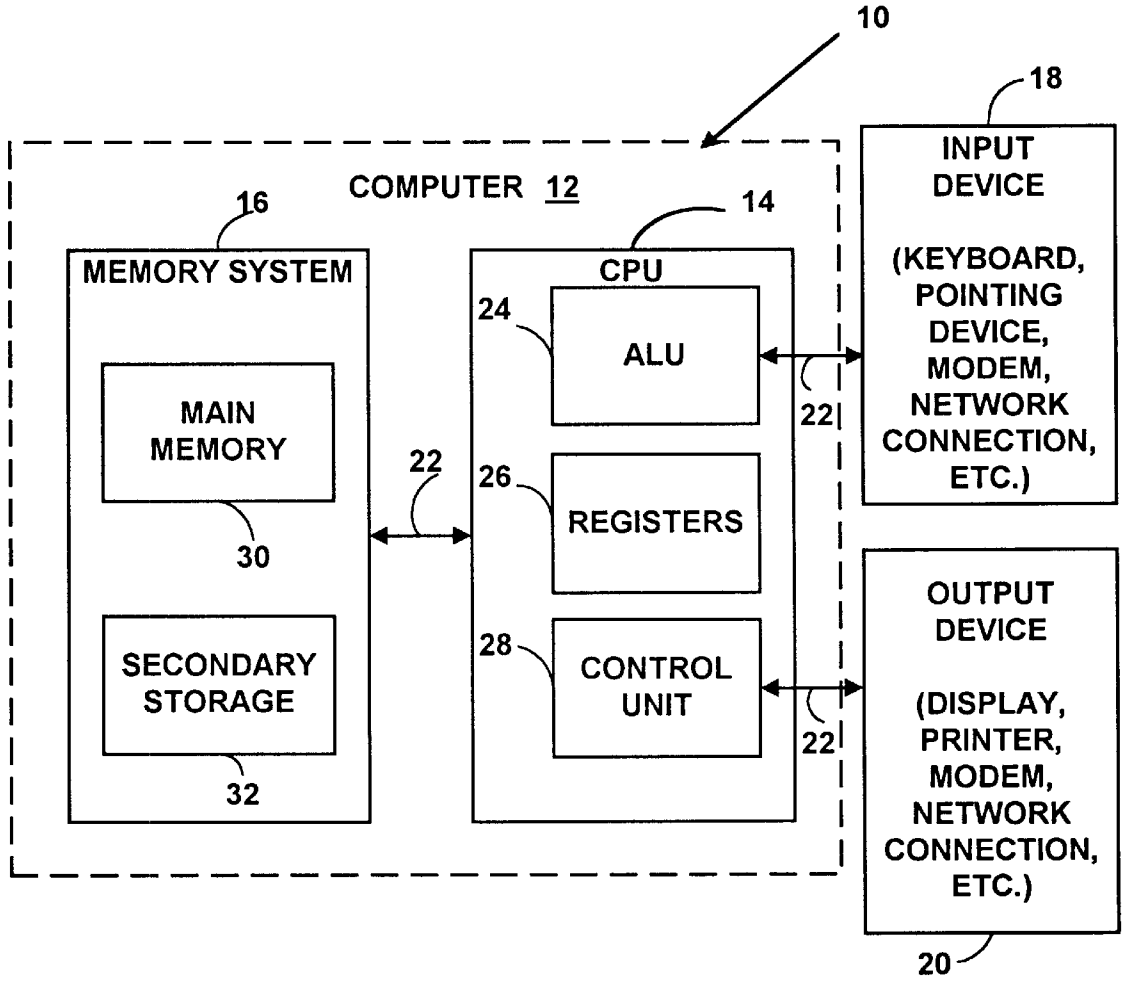

10
18
INPUT DEVICE
(KEYBOARD, POINTING DEVICE, MODEM, NETWORK CONNECTION, ETC.)
COMPUTER 12
16
MEMORY SYSTEM
MAIN MEMORY
30
SECONDARY STORAGE
32
22
14
CPU
24
ALU
26
REGISTERS
28
CONTROL UNIT
22
22
OUTPUT DEVICE
(DISPLAY, PRINTER, MODEM, NETWORK CONNECTION, ETC.)
20

FIG. 2

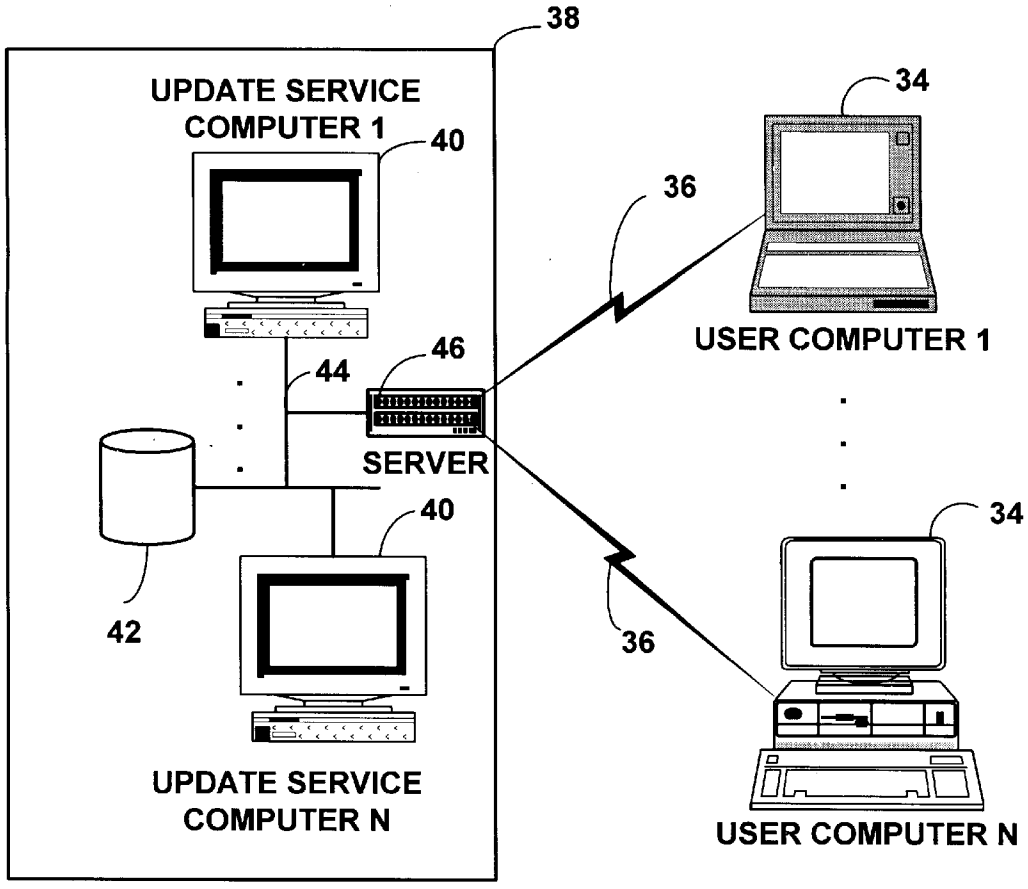

38
UPDATE SERVICE COMPUTER 1
40
34
36
USER COMPUTER 1
46
44
SERVER
40
34
42
36
UPDATE SERVICE COMPUTER N
USER COMPUTER N

FIG. 3

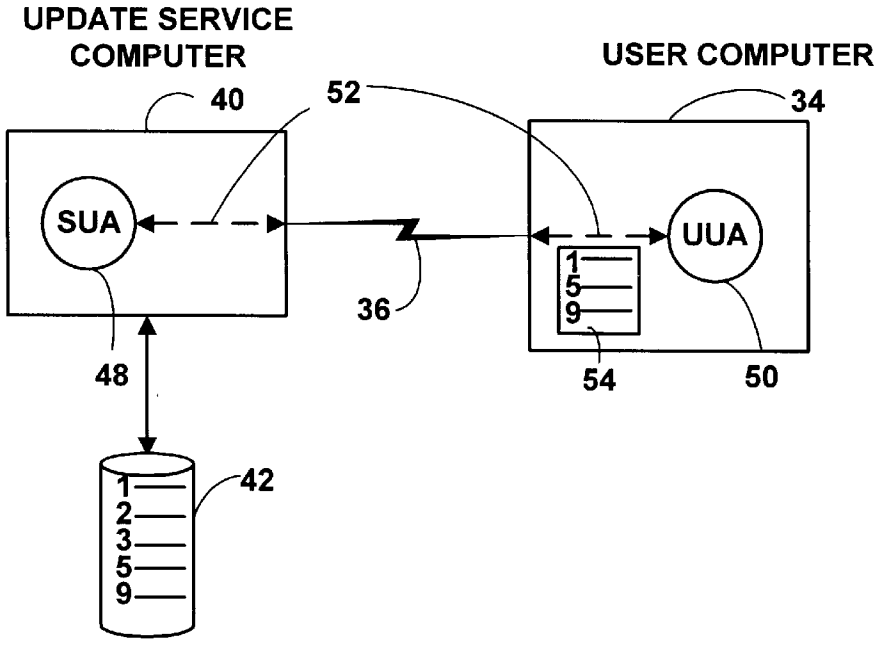
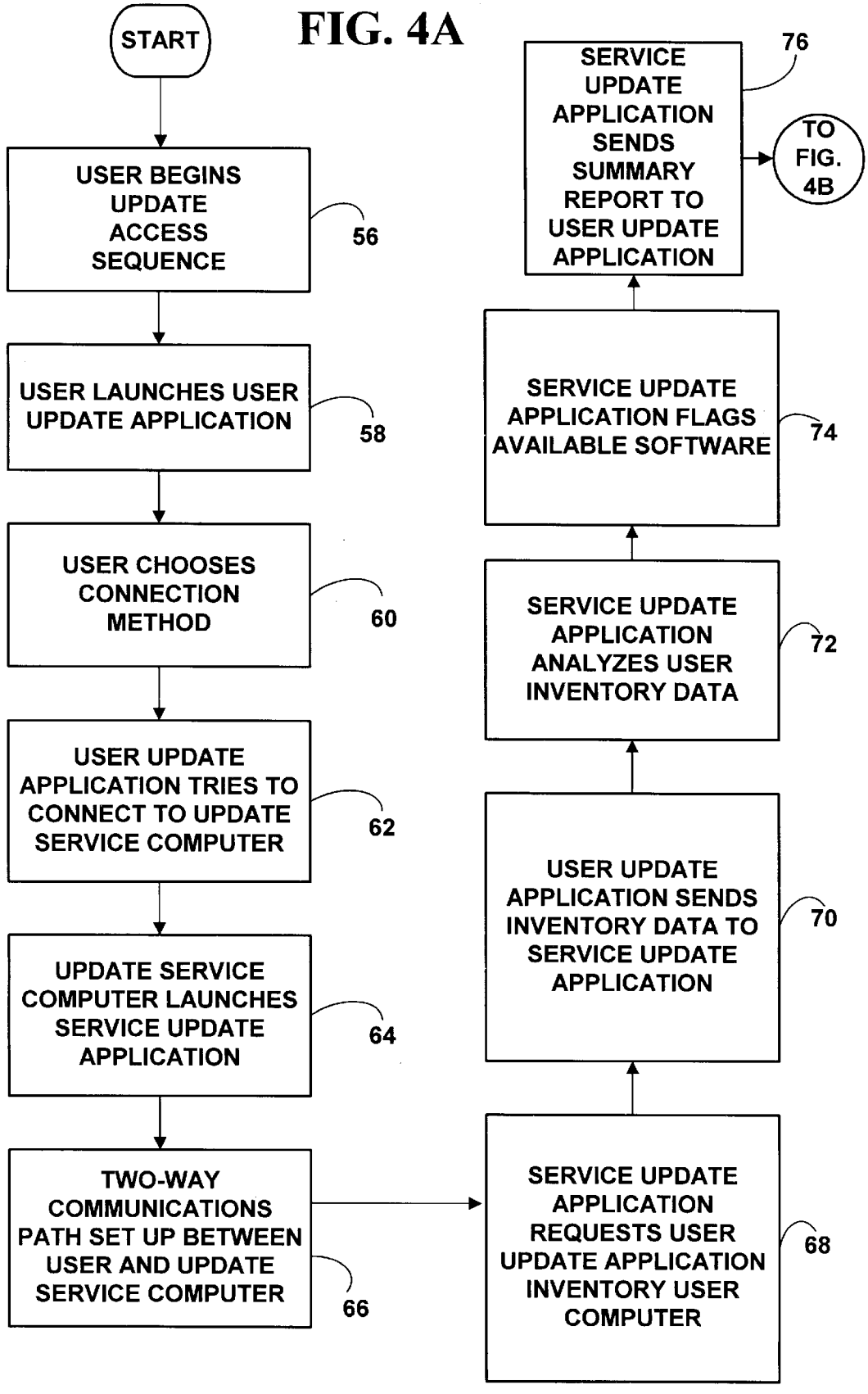

UPDATE SERVICE COMPUTER
USER COMPUTER
40
52
34
SUA
UUA
1
5
9
36
48
54
50
42
1
2
3
5
9

FIG. 4A


START
USER BEGINS UPDATE ACCESS SEQUENCE
56
USER LAUNCHES USER UPDATE APPLICATION
58
USER CHOOSES CONNECTION METHOD
60
USER UPDATE APPLICATION TRIES TO CONNECT TO UPDATE SERVICE COMPUTER
62
UPDATE SERVICE COMPUTER LAUNCHES SERVICE UPDATE APPLICATION
64
TWO-WAY COMMUNICATIONS PATH SET UP BETWEEN USER AND UPDATE SERVICE COMPUTER
66
SERVICE UPDATE APPLICATION REQUESTS USER UPDATE APPLICATION INVENTORY USER COMPUTER
68
USER UPDATE APPLICATION SENDS INVENTORY DATA TO SERVICE UPDATE APPLICATION
70
SERVICE UPDATE APPLICATION ANALYZES USER INVENTORY DATA
72
SERVICE UPDATE APPLICATION FLAGS AVAILABLE SOFTWARE
74
SERVICE UPDATE APPLICATION SENDS SUMMARY REPORT TO USER UPDATE APPLICATION
76
TO FIG. 4B

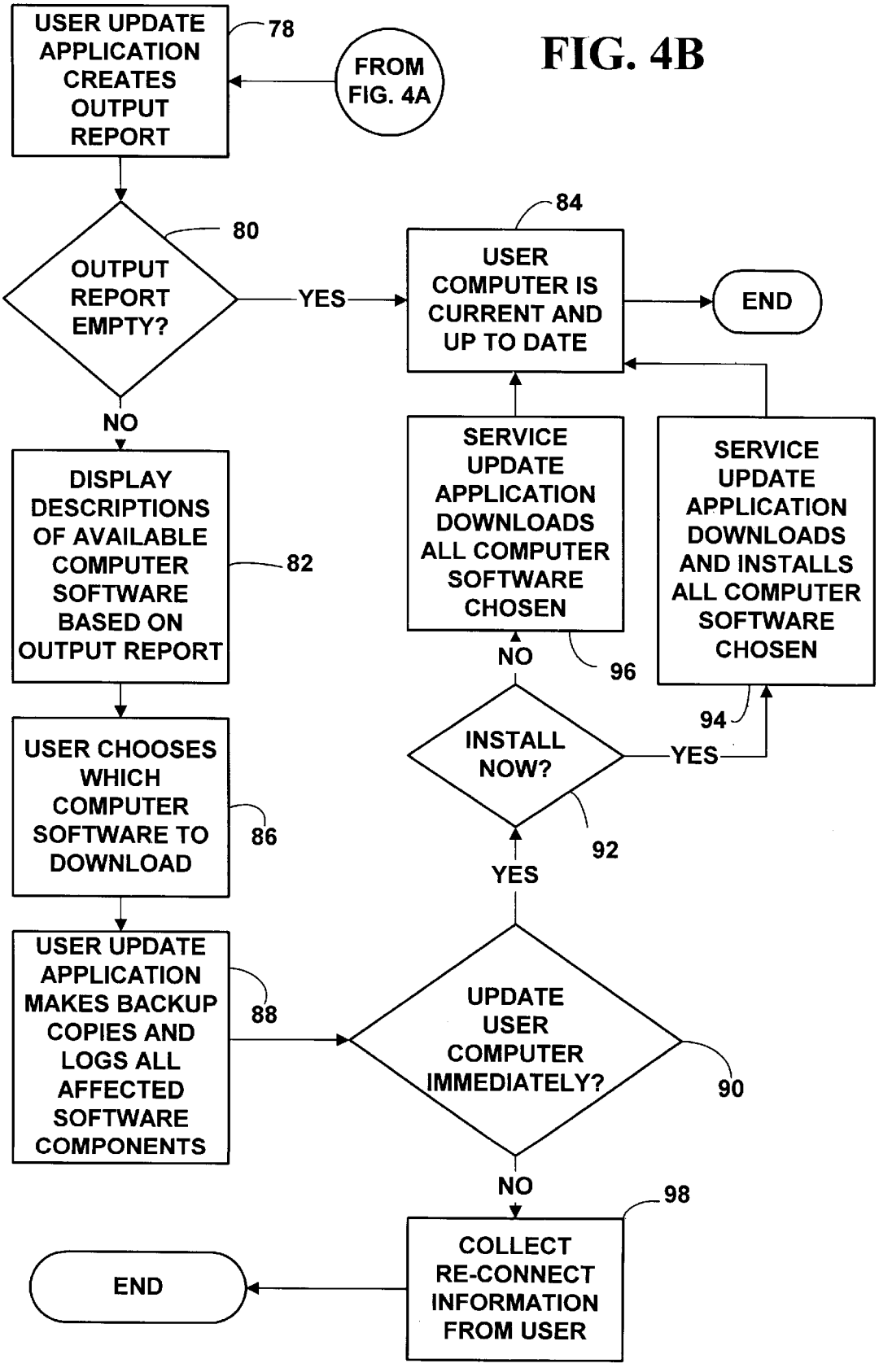
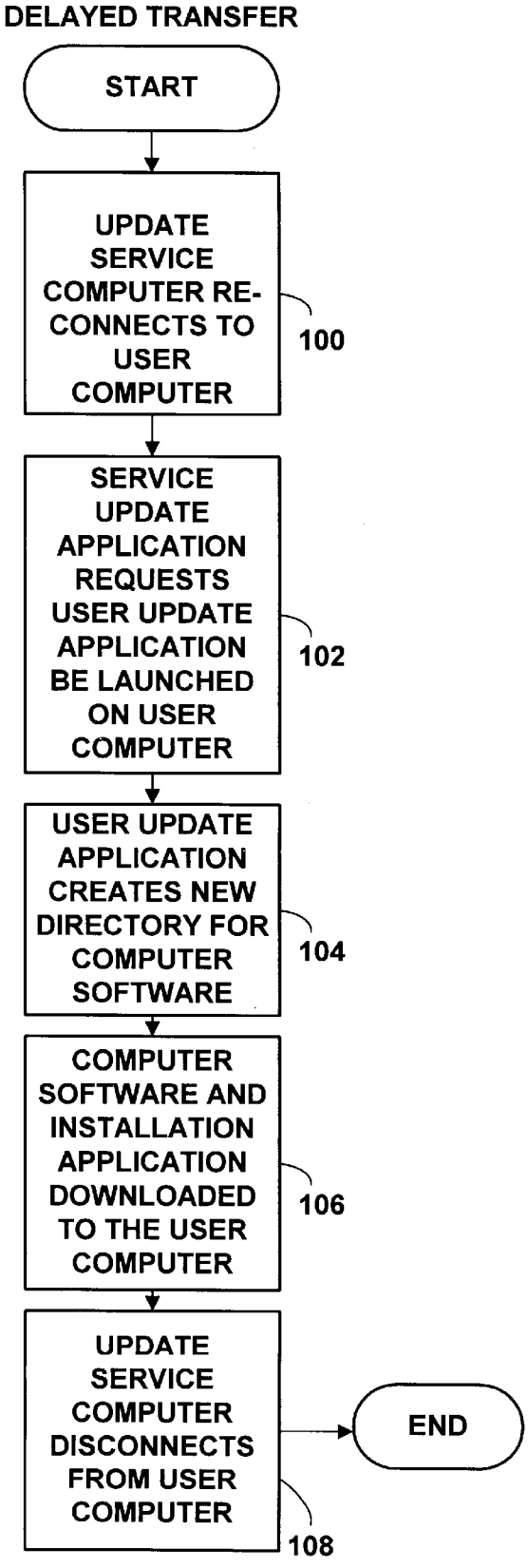

USER UPDATE APPLICATION CREATES OUTPUT REPORT
78
FROM FIG. 4A
OUTPUT REPORT EMPTY?
80
YES
NO
USER COMPUTER IS CURRENT AND UP TO DATE
84
END
DISPLAY DESCRIPTIONS OF AVAILABLE COMPUTER SOFTWARE BASED ON OUTPUT REPORT
82
SERVICE UPDATE APPLICATION DOWNLOADS ALL COMPUTER SOFTWARE CHOSEN
SERVICE UPDATE APPLICATION DOWNLOADS AND INSTALLS ALL COMPUTER SOFTWARE CHOSEN
USER CHOOSES WHICH COMPUTER SOFTWARE TO DOWNLOAD
86
NO
96
INSTALL NOW?
YES
94
92
YES
USER UPDATE APPLICATION MAKES BACKUP COPIES AND LOGS ALL AFFECTED SOFTWARE COMPONENTS
88
UPDATE USER COMPUTER IMMEDIATELY?
90
NO
98
COLLECT RE-CONNECT INFORMATION FROM USER
END

FIG. 4B

FIG. 5


DELAYED TRANSFER
START
UPDATE SERVICE COMPUTER RE-CONNECTS TO USER COMPUTER
100
SERVICE UPDATE APPLICATION REQUESTS USER UPDATE APPLICATION BE LAUNCHED ON USER COMPUTER
102
USER UPDATE APPLICATION CREATES NEW DIRECTORY FOR COMPUTER SOFTWARE
104
COMPUTER SOFTWARE AND INSTALLATION APPLICATION DOWNLOADED TO THE USER COMPUTER
106
UPDATE SERVICE COMPUTER DISCONNECTS FROM USER COMPUTER
108
END

METHOD AND SYSTEM FOR IDENTIFYING AND OBTAINING COMPUTER SOFTWARE FROM A REMOTE COMPUTER

FIELD OF INVENTION

The present invention relates to a system for automatically identifying software that may be appropriate for installation on a computer and for making that software available to that computer. In particular the invention relates to a remote system that inventories software installed on a computer, identifies additional software that may be appropriate for the computer (e.g. patches, fixes, new versions of existing software, new software, etc.), and makes the identified software available to that computer.

BACKGROUND AND SUMMARY OF THE INVENTION

The continual and rapid development of computers, computer software and related technology has revealed many problems with the typical distribution channels for computer software. For example, computer software, the coded instructions that control a computer's operation, are constantly and incrementally being upgraded and improved. The computer hardware and operating system environment on which the computer software is used is continually being changed, which requires additional changes in the computer software (e.g. new device drivers, new operating system calls, etc.).

A computer software developer will typically release an initial version of a software product. Thereafter, as new and improved computers and peripherals are developed, the software product will commonly be upgraded so as to take full advantage of the increased capabilities of the hardware. In addition, a software developer, to remain competitive, will often upgrade the software product to provide new features and functionality.

With the ever increasing pace of advancement in computer related technologies, software developers compete to be the first to offer a new feature or upgrade. As a result, sometimes software products are made available to the public with unknown errors or defects. Similarly, software products that work as intended on a particular computer with a particular configuration, may fail when installed on a different computer having a different configuration (e.g. different hardware, peripherals, operating systems, etc.). Software developers frequently provide fixes for their software products to correct defects that were undetected or unanticipated at the time the software product was released. Fixes are also provided to allow the software product to function correctly on a new computer or with a different operating system environment.

However, it is often difficult for software developers to make upgrades and fixes available to users. This difficulty not only deprives the user of access to the most reliable and up-to-date software products, it can result in lost sales to the software developer and can damage the goodwill and the development of a long term relationship with a customer by releasing a flawed or deficient software product.

Commonly, mass distribution of commercial software products is accomplished by copying the software product onto storage media (e.g. CD-ROMs, floppy disks, magnetic tapes, etc.). To take advantage of economies of scale, typically a large number of copies of the software product are made during the manufacture of a particular software product. Then, the storage media containing the software product is provided to distributors and retailers for sale to

users. However, given the rapid pace of software development, this manner of distribution is frequently insufficient. For example, it is not uncommon that defects are detected and fixes created shortly after a software product is introduced to the public. However, the software products that remain in the distribution chain contain the defect without the fix. This situation is frustrating for users who subsequently purchase the software product that is already obsolete (i.e. because of the defects).

Software can also be distributed over electronic bulletin board systems, the Internet, etc. In such systems, a user connects to the bulletin board and then selects and downloads desired software. Such systems allow for rapid updating of software by simply supplying a new updated version of the software to the bulletin board. However, such systems also require a degree of user sophistication and technical expertise in the selection, downloading and installation of the new software. Moreover, such systems do not provide a user that has already obtained a software product with a simple, automatic way of learning of or obtaining upgrades or fixes for that product. The software provider may also have updated help files and other help utilities about which a user would have no way of knowing.

In accordance with an illustrated embodiment of the present invention, many of the problems associated with obtaining computer software are overcome. A user, with a user computer is allowed to access (e.g. with a modem, an Internet connection, etc.) an update service at a remote location on which is stored a variety of computer software. When a user accesses the remote update service, an update service computer conducts an automatic inventory of the computer software on the user computer. The data collected from the inventory of the user computer software is then used to make comparisons to database entries from a database on the update service computer. The database entries contain information about computer software available on the update service computer. The comparison is conducted to identify software available from the remote update service that might be appropriate for installation on the user computer (i.e. new computer software, new versions of existing computer software, patches or fixes for existing computer software, new help files, etc.). After the comparison is completed, the update service computer makes the computer software stored at the remote update service computer available to the user.

In one aspect of the invention, available computer software can be downloaded from the remote update service computer and installed immediately on the user computer. Another aspect of the invention allows the update service computer to contact the user computer at a later, more convenient time, re-establish two-way communications, then download and install available computer software on the user computer. If a delayed download is requested, the user will provide access information (e.g. phone number, network address, a file of commands to execute to logon the user computer, etc.) to the update service computer which allows the remote update service computer to re-connect to the user computer. The transfer may use an encryption scheme to permit safe transfer of the software to the user computer.

In yet another aspect of the invention, the system will allow a user to purchase the available computer software electronically. The user, for example, provides credit card information, debit card information, an account number to bill, etc. to the update service computer. Secure transaction technology and/or digital signatures are used to safeguard the payment information. After verifying the payment

information, the update service computer permits transfer of the computer software.

The update service has several advantages. A user is automatically provided with information about the available versions of computer software as result of the inventory conducted by the update center computer. If the version of the computer software on the user computer has defects that are known and have been corrected, the user is alerted to this fact and is offered an up-to-date version of the computer software. The user is also alerted to the availability of new computer software, or enhanced versions of existing computer software, and can purchase them electronically. In either case, the most up-to-date versions of computer software are available for downloading to users.

The available versions of the computer software can also be automatically installed on the user computer. Since it is no longer necessary for the user to install the computer software, the incidence of user related installation problems is greatly reduced. It is also not necessary for the user to obtain or save any storage media since the computer software is downloaded directly to the user computer. If the computer software installed on the user computer ever gets corrupted, the user can call the update service (e.g. for some limited number of iterations) and download a new (and up-to-date) copy of the computer software.

In addition to providing benefits for the user, the illustrated embodiment of the invention provides benefits to the developers of the software. The developers of the computer software save support, distribution, and advertising costs. A user who calls the update service automatically obtains up-to-date versions of available computer software, and may never encounter defects which would have been encountered using an earlier, defective version of the computer software. As a result, a user will require less support from the developers of the software will be more satisfied, and be more willing to purchase future versions of computer software. Since the computer software is downloaded to the user computer, the developers of the computer software may save distribution costs as fewer versions of the computer software have to be copied to storage media and distributed. In addition, since the user is also alerted when new computer software, and/or new versions of existing computer software are available, the software developers may also save advertising costs.

The foregoing and other features and advantages of the illustrated embodiment of the present invention will be more readily apparent from the following detailed description, which proceeds with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a computer system used to implement an illustrated embodiment of the present invention.

FIG. **2** is a block diagram showing the update service center and the remote user computers.

FIG. **3** is a block diagram showing the access processes on the user and update service computers.

FIGS. **4A–4B** are a flow chart showing an illustrative sequence which is followed when a user calls the update service.

FIG. **5** is a flow chart showing an illustrative sequence which is followed when the update service re-connects to a user computer.

DETAILED DESCRIPTION OF AN ILLUSTRATED EMBODIMENT

Referring to FIG. **1**, an operating environment for the illustrated embodiment of the present invention is a computer system **10** with a computer **12** that comprises at least one high speed processing unit (CPU) **14**, in conjunction with a memory system **16**, an input device **18**, and an output device **20**. These elements are interconnected by a bus structure **22**.

The illustrated CPU **14** is of familiar design and includes an ALU **24** for performing computations, a collection of registers **26** for temporary storage of data and instructions, and a control unit **28** for controlling operation of the system **10**. Any of a variety of processors, including those from Digital Equipment, Sun, MIPS, IBM, Motorola, NEC, Intel, Cyrix, AMD, Nexgen and others are equally preferred for CPU **14**. Although shown with one CPU **14**, computer system **10** may alternatively include multiple processing units.

The memory system **16** includes main memory **30** and secondary storage **32**. Illustrated main memory **30** is high speed random access memory (RAM) and read only memory (ROM). Main memory **30** can include any additional or alternative high speed memory device or memory circuitry. Secondary storage **32** takes the form of long term storage, such as ROM, optical or magnetic disks, organic memory or any other volatile or non-volatile mass storage system. Those skilled in the art will recognize that memory **16** can comprise a variety and/or combination of alternative components.

The input and output devices **18**, **20** are also familiar. The input device **18** can comprise a keyboard, mouse, pointing device, sound device (e.g. a microphone, etc.), or any other device providing input to the computer system **10**. The output device **20** can comprise a display, a printer, a sound device (e.g. a speaker, etc.), or other device providing output to the computer system **10**. The input/output devices **18**, **20** can also include network connections, modems, or other devices used for communications with other computer systems or devices.

As is familiar to those skilled in the art, the computer system **10** further includes an operating system and at least one application program. The operating system is a set of software which controls the computer system's operation and the allocation of resources. The application program is a set of software that performs a task desired by the user, making use of computer resources made available through the operating system. Both are resident in the illustrated memory system **16**.

In accordance with the practices of persons skilled in the art of computer programming, the present invention is described below with reference to symbolic representations of operations that are performed by computer system **10**, unless indicated otherwise. Such operations are sometimes referred to as being computer-executed. It will be appreciated that the operations which are symbolically represented include the manipulation by CPU **14** of electrical signals representing data bits and the maintenance of data bits at memory locations in memory system **16**, as well as other processing of signals. The memory locations where data bits are maintained are physical locations that have particular electrical, magnetic, optical, or organic properties corresponding to the data bits.

As is shown in FIG. **2**, the illustrated embodiment of the invention consists of one or more user computers **34** which are connected over communications links **36** to an update service center **38**. The update service center consists of one or more second remote computer(s) **40**, one or more communications links **36**, and one or more databases **42**.

The update service center **38** consists of one or more computers **40** (e.g. the computer that was described in FIG.

1) which are capable of simultaneous access by a plurality of user computers. If a plurality of update service computers are used, then the update service computers may be connected by a local area network (LAN) **44** or any other similar connection technology. However, it is also possible for an update service center to have other configurations. For example, a smaller number of larger computers (i.e. a few mainframe, mini, etc. computers) with a number of internal programs or processes running on the larger computers capable of establishing communications links to the user computers. The update service center may also be connected to a remote network (e.g. the Internet) or a remote site (e.g. a satellite) (which is not shown in FIG. **2**). The remote network or remote site allows the update service center to provide a wider variety of computer software than could be stored at the update service center. One or more databases **42** connected to the update center computer(s) **40** are used to store database entries consisting of computer software available on the update service computer(s). The update service computer(s) also contain a plurality of communications links **36** such as telecommunications connections (e.g. modem connections, ISDN connections, ATM connection, frame relay connections, etc.), network connections (e.g. Internet, etc.), satellite connections (e.g. Digital Satellite Services, etc.), wireless connections, two way paging connections, etc., to allow one or more user computers to simultaneously connect to the update service computer(s). The connections are managed by an update server **46**.

After a user computer establishes two-way communications with the update service computer, an inventory of computer software on the user computer is completed without interaction from the user, sent to the update service computer, and compared to database entries on the update service computer. The database entries from the database connected to the update service computer contain information about computer software which is available to a user. After the comparison, the user computer is sent back a summary of available computer software which is displayed for the user. The summary contains information such as the availability of patches and fixes for existing computer software, new versions of existing computer software, and brand new computer software, new help files, etc. The user is then able to make one or more choices from the summary of available computer software, and have the computer software transferred from the update service computer to the user computer. The user may choose to update on the fly, or store update information for future update needs.

As is shown in FIG. **3**, running on the update service computer(s) **40** is one or more service update applications (SUA) **48** that will communicate with a user update application (UUA) **50** on the user computer when the update service is contacted by the a user with a user computer. The user update application **50** is a computer software program that is capable of initiating, establishing and terminating two-way communications with an update service application on the update service computer. The service update application **48** is a computer software program which is also capable of initiating, establishing and terminating two-way communications with a user update application on a user computer.

To access the update service center **38**, a user starts a user update application (UUA) **50** on the user computer **34** to begin the access process. The user update application **50** tries to establish a two-way communications link **36** with an update service computer **40** using a modem, a network connection (e.g. Internet), etc. However, this access can also be completed by a variety of other methods which provide two-way data transfer. As the user update application **48** on the user computer **34** tries to establish a two-way communications link **36** to the update service computer **40**, the update service computer starts a service update application (SUA) **48**. The service update application on the update service computer then tries to establish a two-way communications link to the user update application on the user computer. This is shown by the dashed lines **52** in FIG. **3**. This communications link can be established with a network protocol stack, (e.g. TCP/IP) through sockets, or any other two-way communications technique known in the art.

After establishing a two-way communications link, the service update application conducts an automatic inventory (i.e. without input from the user) of the computer software on the user computer. The data collected during the inventory is sent from the user computer to the remote update service computer. The service update application on the update service computer compares the inventory data collected from the user computer to data stored in a database on the update service computer. The database contains information on available computer software available from the update service. The update service computer then creates a summary and sends the summary to the user computer. The summary **54** is then presented to the user by the user computer. The summary contains information about computer software available on the update service computer such as the availability of patches and fixes for existing computer software, new versions of existing computer software, and brand new computer software, etc. In addition, the availability of agent help files, wizards, inference engine, and other operating system components will be listed in the summary.

The illustrated embodiment of the invention is implemented in the Microsoft Windows 95 operating system by the Microsoft Corporation of Redmond, Wash. using a modem, or an Internet network connection, for access to the update service computer. The invention can likewise be practiced with other operating systems and other access technologies that allow two-way data transfer.

As is shown in the flowchart in FIG. **4A**, a user begins the access sequence (**56**) to an update service by launching a user update application included in the Windows® 95 operating system. However, the user update application can also be any application that is capable of two-way communications, and run under other operating systems. The user update application allows the user computer to establish a two-way communications path for access to the update service computer.

When the user update application starts (**58**), the user is shown optional help information which instructs the user on how to establish a connection between the user computer and the update service computer. The actual connection configuration is completed by allowing a user to choose the appropriate connection method (**60**). For example, the user may choose to establish the connection with a modem. If a modem is chosen, the phone number to dial, modem (e.g. speed, line type, etc.) and communications parameters (e.g. parity, stop bits, etc.) are then configured. If the user chooses to make a network connection (e.g. Internet, etc.) to access the update service, the network address of the update service and other network parameters are configured. A similar sequence would be completed for other connection technologies.

When the user update application attempts to make the desired connection (**60**), the update service computer launches a service update application (**64**). A two-way

communications path (66) is set up between the service update application on the update service computer and the user update application on the user computer. The service update application on the update service computer then requests that the user update application on the user computer conduct an automatic inventory of all computer software installed (68) on the user computer.

In the illustrated system, this inventory is done automatically (i.e. without input from the user), and is completed by assigning the inventory task to a Windows 95 operating system process thread on the user computer. The operating system thread completes the task in the "background" while the user is performing other activities in the "foreground" (e.g. choosing options from the user update application). In reality, there are no real background and foreground processes, just a number of operation system process and process threads which are run for some specified time interval by the operating system. Threads are well known in the art and are used in other operating systems such as Windows NT by Microsoft, and OS/2 by IBM. However, other operating system techniques could also be used to accomplish the inventory on the user computer.

During the inventory, data is collected about all computer software installed on the user computer. Data such as the software title, date, version, file size, file checksum, directory location on the user computer, etc. are collected. After the inventory is complete, the user update application sends (70) the inventory data from the user computer to the service update application on the update service computer. The service update application compares the user inventory data from the user computer to database entries in the computer software database to automatically analyze the computer software stored on the user computer (72). The database connected to the update service computer has entries which contain information about available computer software. The database entries also identify and describe, for example, components of the computer software, including new computer software, patches, fixes, new help files, wizards, inference engines, other operating system components, updates as well as enhancements and new features of existing computer software. The database entries describing new computer software may also include entries describing brand new computer software (i.e. computer software that is newly created, and not previously existing).

Any computer software installed on the user computer which is listed in the database on the update service computer (e.g. out-of-date and/or require a maintenance update, etc.) is flagged as available (74). The user computer may also contain computer software that is not known by the update service. If the user computer contains computer software which is unknown to the update service computer, this computer software is marked as unknown by the update service computer. After the service update application completes the analysis of user computer software, a summary report is sent back to the user computer from the update service computer (76).

In the illustrated system, the user can choose from several update service options. One option may be to check for maintenance updates for all computer software installed on the user computer that is known by the update service. A second option may be to check only specific computer software, or a specific group of computer software stored on the user computer for maintenance updates. For example, if the user wanted to check and see if there were any maintenance updates for a particular word processing program, option two would be selected. A third option may be to check whether there are any new or enhanced versions of computer software available from the update service. A fourth option may be to check only for new versions of specific computer software or groupings of computer software installed on the user computer. A fifth option may be to check the update service computer for information on new computer software (i.e. brand new products, not new or enhanced versions of existing products). A sixth option may be to check only if there are new help files, or other new support data available. This list of options is not intended to be all inclusive, as other options can be added to provide additional update service functionality. Based on user input, the user update application creates an output report (78) (FIG. 4B) based on the option(s) chosen by the user and the summary report created by the service update application. The service update application can also create the output report directly, using default choices with no input at all from a user.

If the output report is not empty (80), a second optional report is created and displayed for the user providing a short description that summarizes the computer software available from the update service (82). This second optional report is used by the user to determine what computer software on the user computer will be updated, if any. If the output report is empty, the computer software on the user computer is current and up-to-date (84), so no further action by either the update service computer or the user computer is required.

If the output report is not empty, then the user is asked to choose which available computer software shown in the output report, if any, will be downloaded and installed on the user computer (86). No software is downloaded without the user's permission. If one or more computer software components (i.e. pieces or parts of the available computer software) are chosen by the user, the user update application is instructed to make backup copies of all of the computer software components on the user computer that will be affected, and create a log for the user documenting which computer software will be replaced (88). The backup copies and the log can be used by the user to restore the original version of the computer software components on the user computer if a need arises to do so.

The user has the option of choosing none, one, or a number of computer software components to download and install. If the list of available computer software to be downloaded and installed is large, the user also has the option of delaying the update to a later time (90). If the user chooses an immediate download, the user is asked if the service update center should also install the computer software chosen by the user (92) after downloading.

If immediate installation is chosen, the service update application on the update service computer downloads the available software to the user computer and installs the software in the proper place (e.g. in the proper directory or subdirectory) on the user computer (94). A log is also created that records what computer software was downloaded to the user computer. If immediate installation is not chosen by the user, the user can save any update information, and continue with other tasks before deciding when to download any software chosen by the user.

If the user chooses a delayed update, the user provides re-connect information (98) that allows the update service computer to re-connect to the user computer at a more convenient time (e.g. midnight, etc.) and complete the downloading and installation at that time.

As part of the re-connect information, the user may create a logon script using an automated macro language to provide the logon sequence to be used, and the directory to be used to download the software chosen by the user. The logon

script and the time the user wishes to have the chosen software downloaded are then sent to the update service computer, and stored in an update service computer database. At the appropriate time chosen by the user, the update service will execute the logon script to re-connect to the user computer, and download the chosen software in the proper directory. An encryption scheme may also be used to permit safe transfer of the software to the user computer.

The user also has the option of choosing a logon method different then the one they are currently connected to the update service center with. For example, if a user is connected to the update service center with a modem, the user may choose to have the chosen software downloaded at a later time using a network connection (e.g. Internet, etc.) However, the user's computer must be capable of accepting software with a different connection method.

To allow a re-connection using a modem, the user would enter the phone number of the phone line attached to the user computer and send this information to the update service computer. The user would leave the user computer and modem on, and set the communications software in an answer mode to answer any incoming calls. For a network re-connection, the user would provide the update service computer the user network address and set the network software in a host mode to process any network connection attempts.

The delayed downloading is illustrated in the flow chart in FIG. **5**. To complete the delayed downloading, the update service computer launches a service update application that tries to re-connect (**100**) to the user computer. The update service application will use the information provided at an earlier time by a user (e.g. modem logon information, network logon information, a logon script, etc.). If the connection is successful, a service update application on the update service computer asks the user computer to launch a user update application to re-establish a two-way communications path (**102**). The re-connect to the user computer may be completed using a different access method than was used during the original user computer-update service computer connection. For example, the update service computer may request a digital satellite system re-connect to the user computer instead of the update service computer. A different access method is typically chosen to provide the most efficient and greatest bandwidth data transfer between the update service computer and the user computer.

After establishing a new two-way communications path, the user update application creates a new directory (**104**) on the user computer, where the computer software is transferred and stored (**106**). A log is also created to document what available computer software was transferred to the user computer. Included with the downloaded computer software is an installation application that will be used later by the user to install the computer software. When the transfers are complete, the update service computer terminates the connection to the user computer (**108**). An encryption scheme may also be used to permit safe automated transfer of the software to the user computer.

When the user is ready to install the computer software (e.g. the next morning if the computer software was transferred and installed in the middle of the night), the user simply launches the installation application supplied by update service computer.

Leaving the user an installation application to execute is an added safety and security measure for both the user and the update service. The user computer is not updated unless the user personally starts the installation process. However, the user can also choose to have the computer software automatically installed by the update service when it re-connects to the user computer (but, user permission is always obtained and recorded first). In this case, the installation application is not downloaded to the user computer. However a log is created so a user can determine what available computer software was downloaded and installed.

As was described above, the third, fourth, and fifth options allow a user to check for new versions of existing computer software, or new computer software available from the update service. If a new version of existing computer software, or new computer software is available, the user is asked if they wish to purchase the computer software. If so, the appropriate fee is requested from the user. The user can pay the fee electronically by transmitting credit card information, debit card information, billing account information, etc. to the update service computer from the user computer. Digital signatures, secure transaction technology, or an encryption scheme may also be used to collect payment information from the user. Once the fee information is collected by the update service computer and is verified, the user can choose between immediate or delayed downloading of the new, or new version of the computer software following steps (**88–98**) (FIG. **4**B) described above.

Since new versions of computer software are typically very large, the user will be informed that a delayed installation is probably most efficient for the user. If a delayed installation of a new product is chosen, the update service computer will then re-connect to the user computer at a later time and download the new version of the computer software as was shown in FIG. **5**. In the illustrated embodiment, the update service uses a digital satellite service link, or some other higher bandwidth connection to transfer the computer software to the user computer whenever possible.

If the user chooses not to pay for a new version of computer software when the update service is called, additional data from which the user can obtain more information on the new computer software is displayed. For example, the information may contain a summary of the features of the new computer software and the information may also contain a list of retail outlets close to the user where the user may then purchase a new version of computer software on storage media if desired. Some users may prefer to obtain the computer software on storage media and call the update service to obtain up-to-date versions of the computer software.

With automatic downloading and installation of computer software from the update service, the user is relieved from the burden of obtaining computer software (e.g. on storage media, by downloading from a bulletin board or on-line service, etc.), and installing the computer software on the user computer. Once a user purchases computer software, periodic calls to the update service will keep the user current and up-to-date.

It should be understood that the programs, processes, or methods described herein are not related or limited to any particular type of computer apparatus, unless indicated otherwise. Various types of general purpose or specialized computer apparatus may be used with or perform operations in accordance with the teachings described herein.

Having illustrated and described the principles of the present invention in an illustrated embodiment, it should be apparent to those skilled in the art that the embodiment can be modified in arrangement and detail without departing from such principles. For example, elements of the illus-

trated embodiment shown in software may be implemented in hardware and vice versa. Similarly, hardware and software components can be interchanged with other components providing the same functionality.

In view of the wide variety of embodiments to which the principles of this invention can be applied, it should be understood that the illustrated embodiments are exemplary only, and should not be taken as limiting the scope of my invention. Rather, I claim as my invention all such embodiments as come within the scope and spirit of the following claims and equivalents thereto.

I claim:

1. In a computer system having a first computer in communication with a remote second computer, the second computer having access to a database identifying software remotely available to the first computer, wherein at least one item in the database identifies software installable on the first computer, a computer implemented method for identifying computer software available for installation on the first computer, the method comprising, at the second computer:

retrieving from the first computer to the second computer an inventory identifying at least certain computer software installed on the first computer;

comparing the inventory of computer software with the database to identify computer software available to the first computer and not installed on the first computer;

preparing for presentation at the first computer software information indicating software available to the first computer and not installed on the first computer; and

sending the software information to the first computer, said information including an alert about a defect in software on the first computer correctable by software available to the first computer and not installed thereon.

2. A computer readable medium having computer executable instructions for performing the method recited in claim **1**.

3. The method of claim **1** wherein the software information comprises purchase information associated with software available to the first computer and not installed on the first computer.

4. The method of claim **1** wherein the software information comprises a description summarizing features of new software available to the first computer and not installed on the first computer.

5. The method of claim **1** wherein the software information comprises a software title, a software date, a software version, a software file size, a software location, and a description of software functionality.

6. The method of claim **1** wherein the second computer is in communication with a third remote computer, the third computer having a second database containing database entries listing a plurality of computer software available to the first computer, the method further comprising:

consulting the third remote computer to compare the inventory with the second database, thereby identifying computer software available to the first computer and not installed on the first computer.

7. The method of claim **1** further comprising:

receiving at the second computer a user software selection indicative of desired software; and

transferring from the second computer to the first computer software indicated by the user software selection.

8. The method of claim **7** further comprising:

collecting payment information from the first computer to pay for the software indicated by the user software selection.

9. The method of claim **7** wherein the software indicated by the user software selection is transferred to the first computer in an encrypted format over a public network.

10. In a computer system having a first computer in communication with a remote second computer, the second computer having access to a database identifying software available to the first computer, wherein at least one item in the database identifies software installable on the first computer, a computer implemented method for identifying computer software available for installation on the first computer, the method comprising, at the first computer:

conducting an inventory of the first computer, the inventory identifying at least certain computer software installed on the first computer;

sending from the first computer to the second computer the inventory for comparison to the database;

receiving from the second computer a software summary indicating software available to the second computer and not installed on the first computer, said software summary including an alert about a defect in software on the first computer correctable by software available to the first computer but not installed thereon; and

presenting the software summary on the first computer.

11. A computer readable medium having computer executable instructions for performing the method recited in claim **10**.

12. The method of claim **10** wherein the database identifies at least one new software product.

13. The method of claim **10** further comprising:

receiving at the first computer a selected category for restricting presented summary information; and

restricting the software summary to software in the selected category.

14. The method of claim **10** further comprising:

receiving at the first computer a user selection from the software summary, the user selection indicating desired software;

backing up software components on the first computer affected by the desired software;

downloading the desired software to the first computer; and

installing the desired software on the first computer.

15. The method of claim **14** wherein the desired software is downloaded to the first computer with a digital signature over a public network.

16. The method of claim **10** further comprising:

receiving at the first computer a user selection from the software summary, the user selection indicating desired software; and

downloading the desired software to the first computer.

17. In a computer system having a user computer in communication with a remote update service computer having access to a database identifying software available to the update service computer, wherein at least one item in the database identifies software installable on the user computer, a computer implemented method for transferring computer software to the user computer, the method comprising:

establishing a first communications session between the user computer and the update service computer;

collecting an inventory during the first session from the user computer to identify at least certain computer software installed on the user computer;

comparing the inventory of computer software on the user computer with the database to identify software avail-

able to the update service computer and not installed on the user computer;

presenting at the user computer software information indicating software available to the update service computer and not installed on the user computer, said information including an alert about a defect in software on the user computer correctable by software available to the user computer and not installed thereon;

receiving a selection of software from the software information during the first session;

saving a list indicating the software selected;

during the first session, retrieving reconnection information to the update service computer from the user computer for establishing a second communications session with the user computer;

terminating the first communications session;

establishing a second communications session between the user computer and the update service computer using the reconnection information; and

transferring during the second communications session software indicated by the saved list from the update service computer to the user computer.

18. The method of claim **17** wherein the second communications session is scheduled by a user to automatically take place at a time more convenient for transferring software than the first.

19. The method of claim **17** wherein the reconnection information comprises computer software billing information collected from the user computer using secure transaction technology.

20. The method of claim **17** wherein the reconnection information comprises a list of commands executable to establish the second communications session.

21. The method of claim **17** wherein the second communications session is completed using an access method different from the first.

22. In a computer, a software delivery system for providing software to a remote computer, the delivery system comprising:

a database containing entries indicative of software available to the remote computer;

an inventory collector operable for receiving from the remote computer an inventory of software indicating at least certain software installed at the remote computer;

a comparer operable for identifying software in the database and not in the inventory; and

a report generator operable for generating a summary of the software in the database and not in the inventory for presentation to a user at the remote computer, the summary including an alert about a defect in software on the remote computer correctable by software available to the remote computer and not installed thereon, the report generator further operable for sending the summary to the remote computer.

23. The delivery system of claim **22** further comprising:

a selection receiver operable for receiving a user selection from the summary;

a software collection comprising software indicated by the database and installable at the first computer; and

a transferor operable for transferring computer software in the collection indicated by the user selection to the remote computer.

24. A method of updating software earlier installed on a first computer from a library of software stored on a second computer, comprising:

identifying software earlier installed on the first computer;

identifying to a user of the first computer a software update that is available on the second computer corresponding to software identified as earlier installed on the first computer, and alerting the user to a defect in the earlier installed software that would be correctable by installation of said software update; and

in response to user authorization, sending said software update from the second computer to the first computer.

* * * * *

Exhibit I

US005941947A

United States Patent [19]

Brown et al.

[11] Patent Number: 5,941,947

[45] Date of Patent: Aug. 24, 1999

[54] **SYSTEM AND METHOD FOR CONTROLLING ACCESS TO DATA ENTITIES IN A COMPUTER NETWORK**

[75] Inventors: **Ross M. Brown**, Bellvue; **Richard G. Greenberg**, Redmond, both of Wash.

[73] Assignee: **Microsoft Corporation**, Redmond, Wash.

[21] Appl. No.: **08/516,573**

[22] Filed: **Aug. 18, 1995**

[51] **Int. Cl.**[6] **G06F 17/00**

[52] **U.S. Cl.** **709/225**

[58] **Field of Search** 395/200.55, 200.56, 395/200.47, 200.48, 200.49, 186, 187.01, 188.01; 380/23, 24, 25; 709/225

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,184,200	1/1980	Wagner et al.	395/188
4,280,176	7/1981	Tan	395/188
4,432,057	2/1984	Daniell et al. .	
4,493,024	1/1985	Baxter, II et al.	395/188
4,799,153	1/1989	Hann et al.	380/25
4,799,156	1/1989	Shavit et al. .	
4,800,488	1/1989	Agrawal et al. .	
4,858,117	8/1989	Dichiara et al.	395/188
4,899,136	2/1990	Beard et al. .	
4,914,571	4/1990	Baratz et al. .	
5,079,765	1/1992	Nakamura .	
5,113,499	5/1992	Ankney et al.	395/325
5,140,689	8/1992	Kobayashi .	
5,151,989	9/1992	Johnson et al. .	
5,187,790	2/1993	Fast et al. .	
5,247,676	9/1993	Ozur et al. .	
5,257,369	10/1993	Skeen et al. .	
5,265,250	11/1993	Andrade et al. .	
5,291,597	3/1994	Shorter et al. .	
5,307,490	4/1994	Davidson et al. .	
5,321,841	6/1994	East et al. .	
5,329,619	7/1994	Page et al. .	
5,341,477	8/1994	Pitkin et al. .	
5,347,632	9/1994	Filepp et al. .	
5,355,497	10/1994	Cohen-Levy .	
5,367,621	11/1994	Cohen et al. .	
5,371,852	12/1994	Attanasio .	
5,388,255	2/1995	Pytlik et al. .	
5,396,626	3/1995	Nguyen .	

(List continued on next page.)

OTHER PUBLICATIONS

Operating System Concepts, Fourth Edition, Abraham Silberschatz and Peter B. Galvin, pp. 361–380, 431–457, ©1994.

Inside Windows NT, Helen Custer Foreword by David N. Cutler, The Object Manager and Object Security, Chapter Three, pp. 49–81. ©1993.

So . . . Just What is this First Class Thing Anyway? (visited Oct. 10, 1995) <http://orion.edmonds.wednet.edu/ESD/FC/AboutFC.html>.

Colton, Malcolm, "Replicated Data in a Distributed Environment," *IEEE* (1993).

(List continued on next page.)

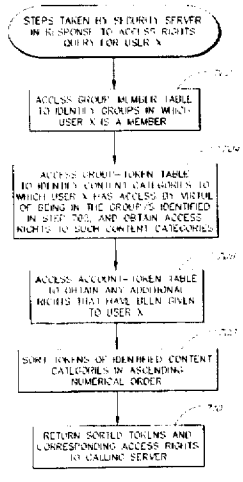
Primary Examiner—Ellis B. Ramirez

Attorney, Agent, or Firm—Leydig, Voit & Mayer, Ltd.

[57] **ABSTRACT**

Access rights of users of a computer network with respect to data entities are specified by a relational database stored on one or more security servers. Application servers on the network that provide user access to the data entities generate queries to the relational database in order to obtain access rights lists of specific users. An access rights cache on each application server caches the access rights lists of the users that are connected to the respective application server, so that user access rights to specific data entities can rapidly be determined. Each user-specific access rights list includes a series of category identifiers plus a series of access rights values. The category identifiers specify categories of data entities to which the user has access, and the access rights values specify privilege levels of the users with respect to the corresponding data entity categories. The privilege levels are converted into specific access capabilities by application programs running on the application servers.

66 Claims, 10 Drawing Sheets



U.S. PATENT DOCUMENTS

5,423,003	6/1995	Berteau .
5,434,994	7/1995	Shaheen et al. .
5,444,848	8/1995	Johnson et al. .
5,455,932	10/1995	Major et al. .
5,463,625	10/1995	Yasrebi .
5,473,599	12/1995	Li et al. .
5,475,819	12/1995	Miller et al. .
5,481,720	1/1996	Loucks et al. .
5,483,652	1/1996	Sudama et al. .
5,490,270	2/1996	Devarakonda et al. .
5,491,800	2/1996	Goldsmith et al. .
5,491,817	2/1996	Gopal et al. .
5,491,820	2/1996	Belove et al. .
5,497,463	3/1996	Stein et al. .
5,499,342	3/1996	Kurihara et al. .
5,500,929	3/1996	Dickinson .
5,513,314	4/1996	Kadasamy et al. .
5,526,491	6/1996	Wei .
5,530,852	6/1996	Meske et al. .
5,544,313	8/1996	Shachanai et al. .
5,544,327	8/1996	Dan et al. .
5,548,724	8/1996	Akizawa et al. .
5,548,726	8/1996	Pettus .
5,551,508	9/1996	Pettus et al. .
5,553,239	9/1996	Heath et al. .
5,553,242	9/1996	Russell et al. .
5,559,969	9/1996	Jennings .
5,564,043	10/1996	Siefert .
5,572,643	11/1996	Judson .
5,581,753	12/1996	Terry et al. .
5,592,611	1/1997	Midgely et al. .
5,596,579	1/1997	Yasrebi .
5,596,744	1/1997	Dao .
5,608,865	3/1997	Midgely et al. .
5,608,903	3/1997	Prasad et al. .
5,617,568	4/1997	Ault et al. .
5,617,570	4/1997	Russell et al. .
5,619,632	4/1997	Lamping et al. .
5,650,994	7/1997	Daley .
5,666,519	9/1997	Hayden .
5,675,723	10/1997	Ekrot et al. .
5,675,796	10/1997	Hodges et al. .
5,696,895	12/1997	Hemphill .
5,774,668	6/1998	Choquire et al. .

OTHER PUBLICATIONS

Coulouris et al., "Distributed Transactions," Chapter 14 of *Distributed Systems Concepts and Design 2nd Ed.*, 409–421 (1994).

Cox, John, "Sybase Server to Add Complexity User for Challenge with Data Replication," *Communication* No. 483 (1993).

Eckerson, Wayne, "Users Give Green Light for Replication," *Network World* (Jul. 19, 1993).

Edelstein, Herb, "The Challenge of Replication," *DBMS* vol. 8, No. 4, 68 (Apr. 1995).

Edelstein, Herb, "Microsoft and Sybase are Adding their Unique Touches to SQI Servers," *Information Week*, No. 528, 62 (1995).

Edelstein, Herb, "Replicating Data," *DBMS* vol. 6, No. 6, 59 (Jun. 1993).

Gouhle, Michael, "RDBMS Server Choice Gets Tougher," *Network World*, 52 (May 23, 1994).

Heylighen, Francis, "World–Wide Web: A Distributed Hypermedia Paradigm for Global Networking," *Proceedings of the SHARE Europe Spring Conference*, 355–368 (1994).

International Telecommunications Union, *CCITT Blue Book vol. VIII Data Communication Networks Directory*, 3–18 (1989).

King, Adrian, "The User Interface and the Shell," *Inside Windows 95*, Chapter 5 (1994).

Pallatlo, John, "Sybase Lays Out Blue Print for Client/Server Networks," *PC Week*, vol. 9, No. 461, 6 (1992).

PR Newswire Association, Inc., "America On–line Publicly Previews World Wide Web Browser," *Financial News Section* (May 9, 1995).

Quereshi, "The Effect of Workload on the Performance and Availability of Voting Algorithms," *IEEE* (1995).

Rexford, Jennifer, "Window Consistent Replication for Real–Time Applications," *IEEE* (1994).

Richman, Dan, "Sybase to Enhance RDBMS," *Open System Today*, No. 111 (1992).

Terry, Douglas, "Session Guarantees for Weekly Consistent Replicated Data," *IEEE* (1994).

Wang, Yongdong, "Data Replication in a Distributed Heterogenous Database Environment," *IEEE* (1994).

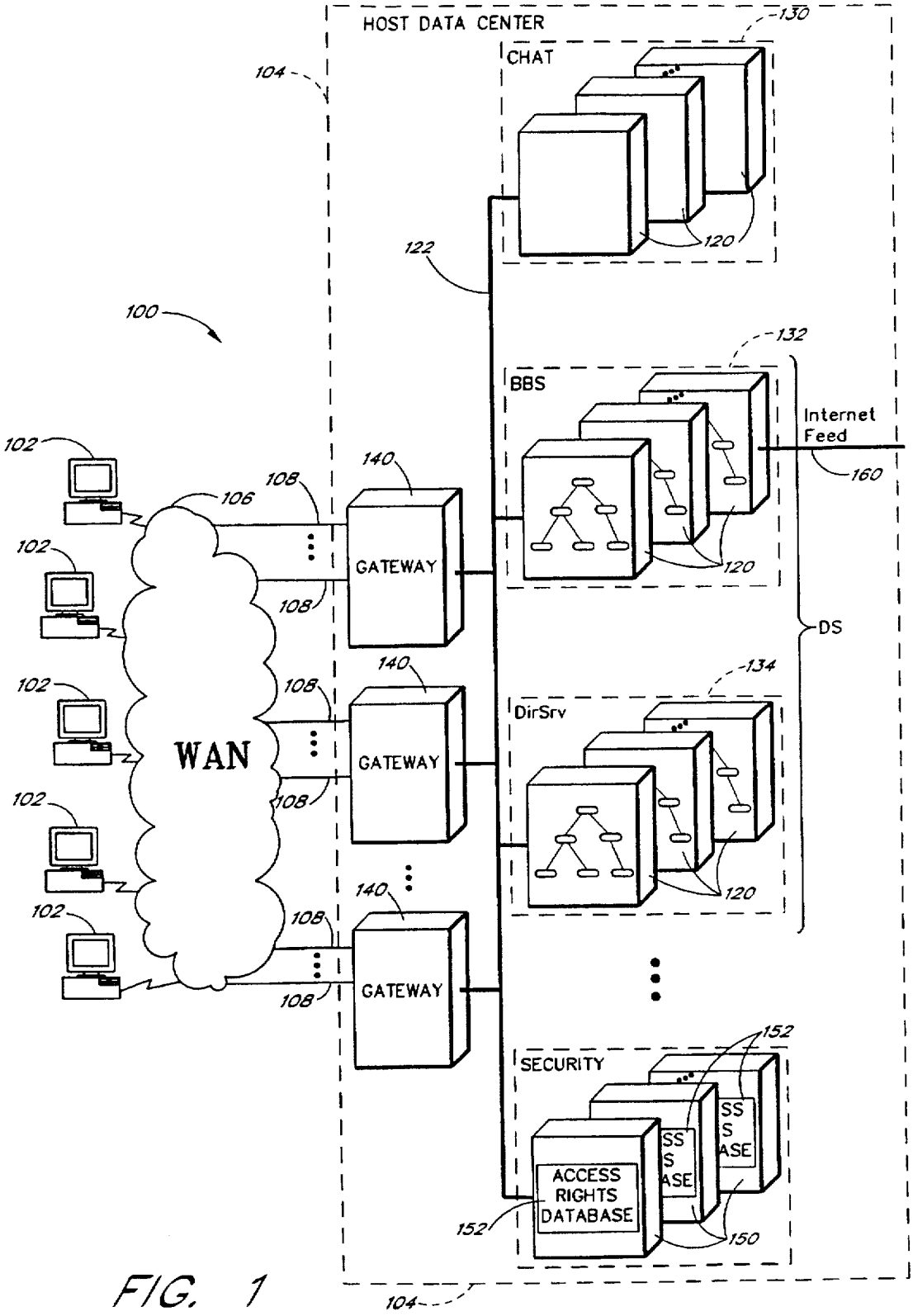

HOST DATA CENTER
130
CHAT
104
120
122
100
132
BBS
Internet Feed
160
102
140
108
106
GATEWAY
120
DS
WAN
134
DirSrv
152
SECURITY
SS S ASE
ACCESS RIGHTS DATABASE
150

FIG. 1

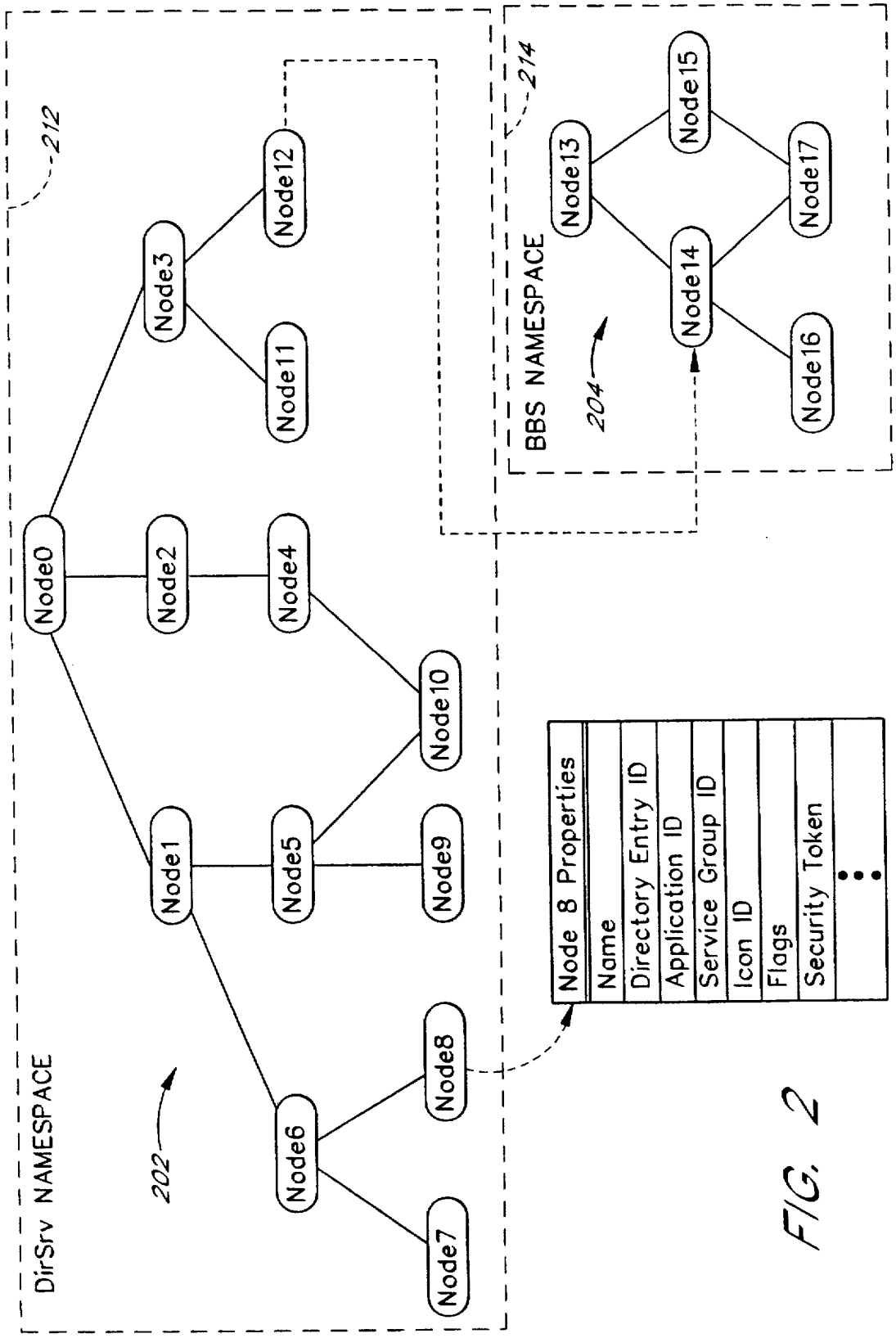

DirSrv NAMESPACE
212
202
Node0
Node1
Node2
Node3
Node4
Node5
Node6
Node7
Node8
Node9
Node10
Node11
Node12
BBS NAMESPACE
214
204
Node13
Node14
Node15
Node16
Node17
Node 8 Properties
Name
Directory Entry ID
Application ID
Service Group ID
Icon ID
Flags
Security Token

FIG. 2

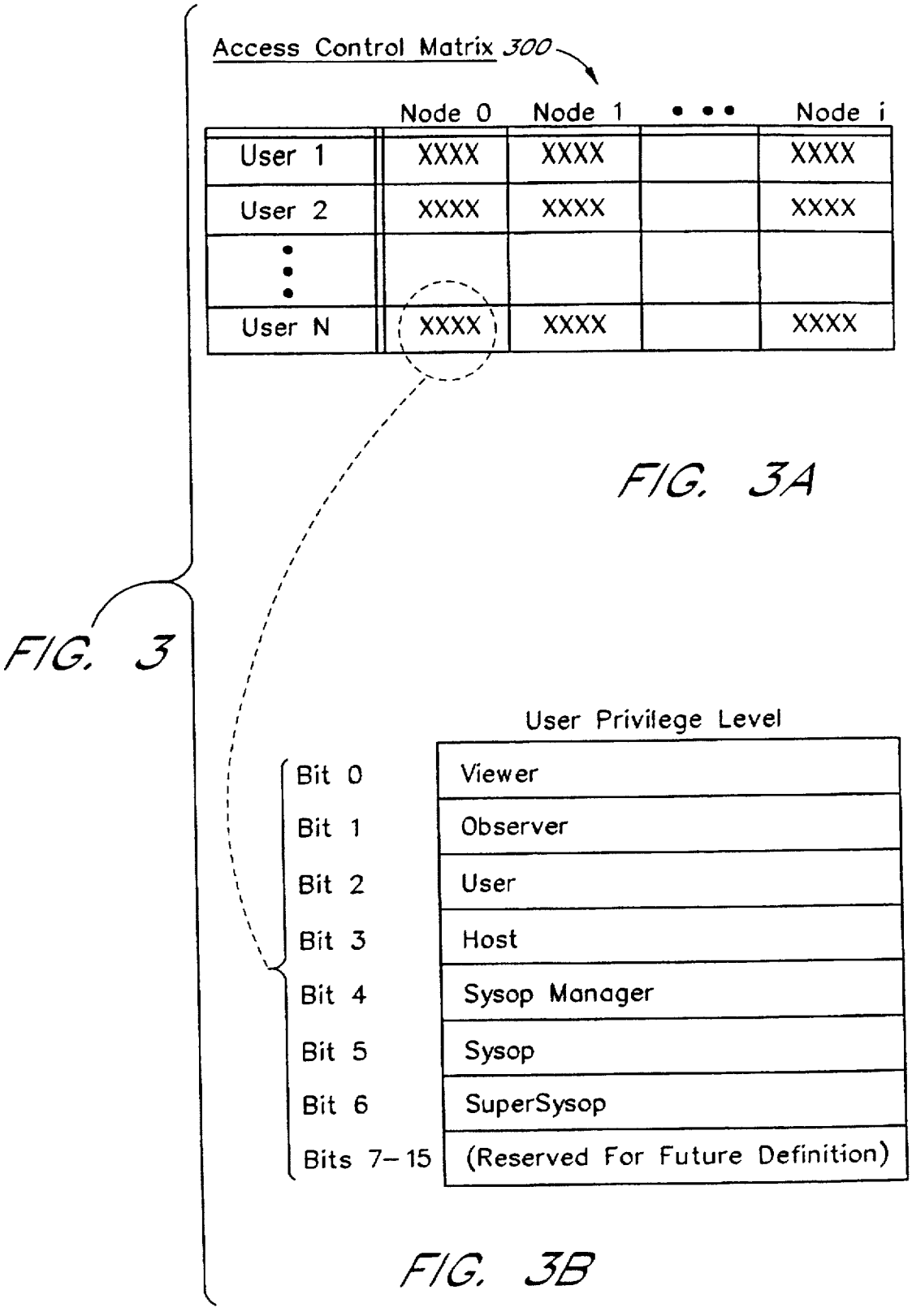


FIG. 3A

FIG. 3B

FIG. 3

300'

	Token 1	Token 2	• • •	Token j
User 1	XXXX	XXXX		XXXX
User 2	XXXX	XXXX		XXXX
• • •				
User N	XXXX	XXXX		XXXX

FIG. 4A

Security Token	Name (Content Category)
1	Internal Public
2	Internal 18–and–Older
3	Internet Public
4	Internet 18–and–Older
• • •	
100	Corporation X Beta Test Data
101	Family and Friends for Brown Family
• • •	

FIG. 4B

300''

	Token 1	Token 2	•••	Token j
Group 1	XXXX	XXXX		XXXX
Group 2	XXXX	XXXX		XXXX
⋮				
Group X	XXXX	XXXX		XXXX
User A	XXXX	XXXX		XXXX
User B	XXXX	XXXX		XXXX
⋮				

502

504

FIG. 5A

Group ID	Group
1	Everyone
2	AllSysops
3	SuperSysops
4	Guest
5	Registration/Sign-Up
6	18-and-Older Access
⋮	

FIG. 5B

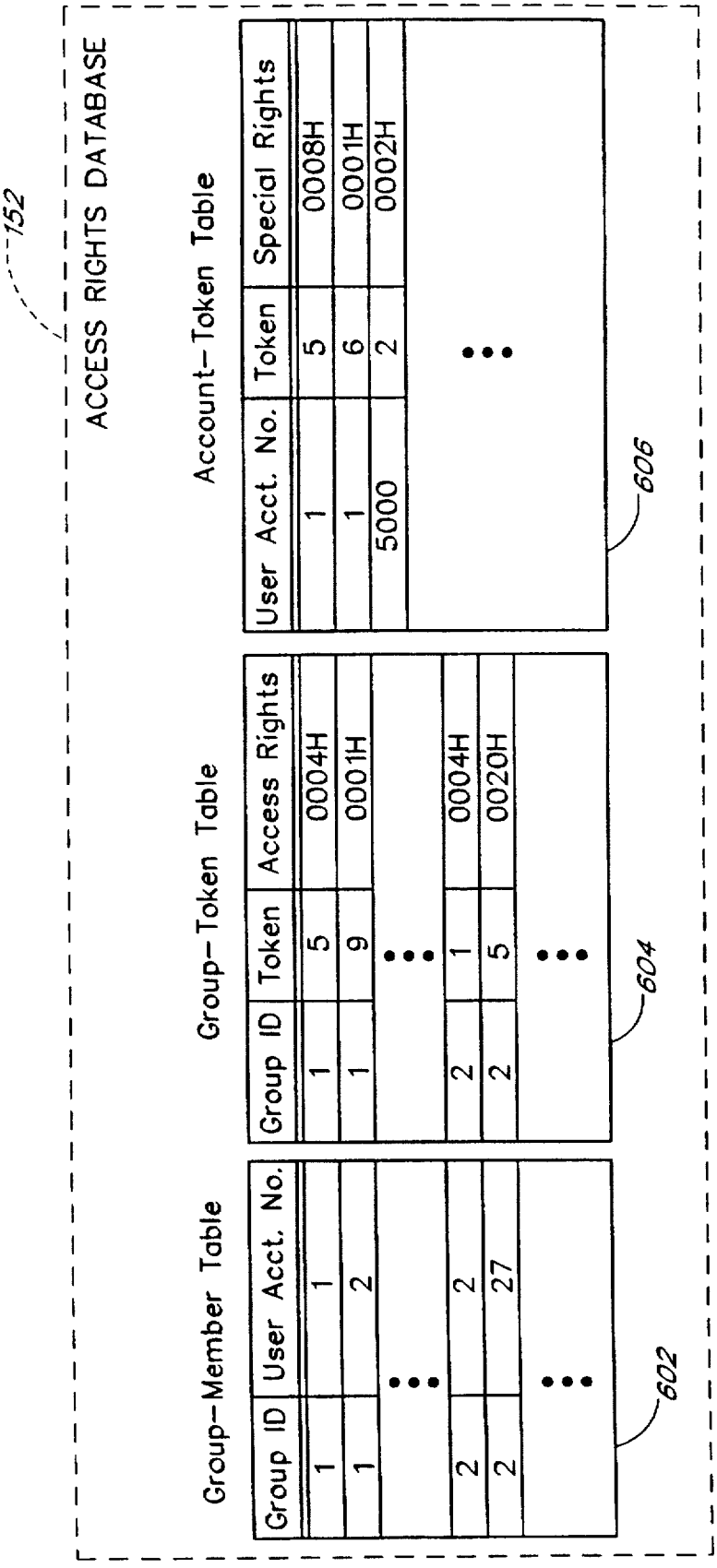

152
ACCESS RIGHTS DATABASE
Group–Member Table
Group ID
User Acct. No.
602
Group–Token Table
Group ID
Token
Access Rights
604
Account–Token Table
User Acct. No.
Token
Special Rights
606

FIG. 6

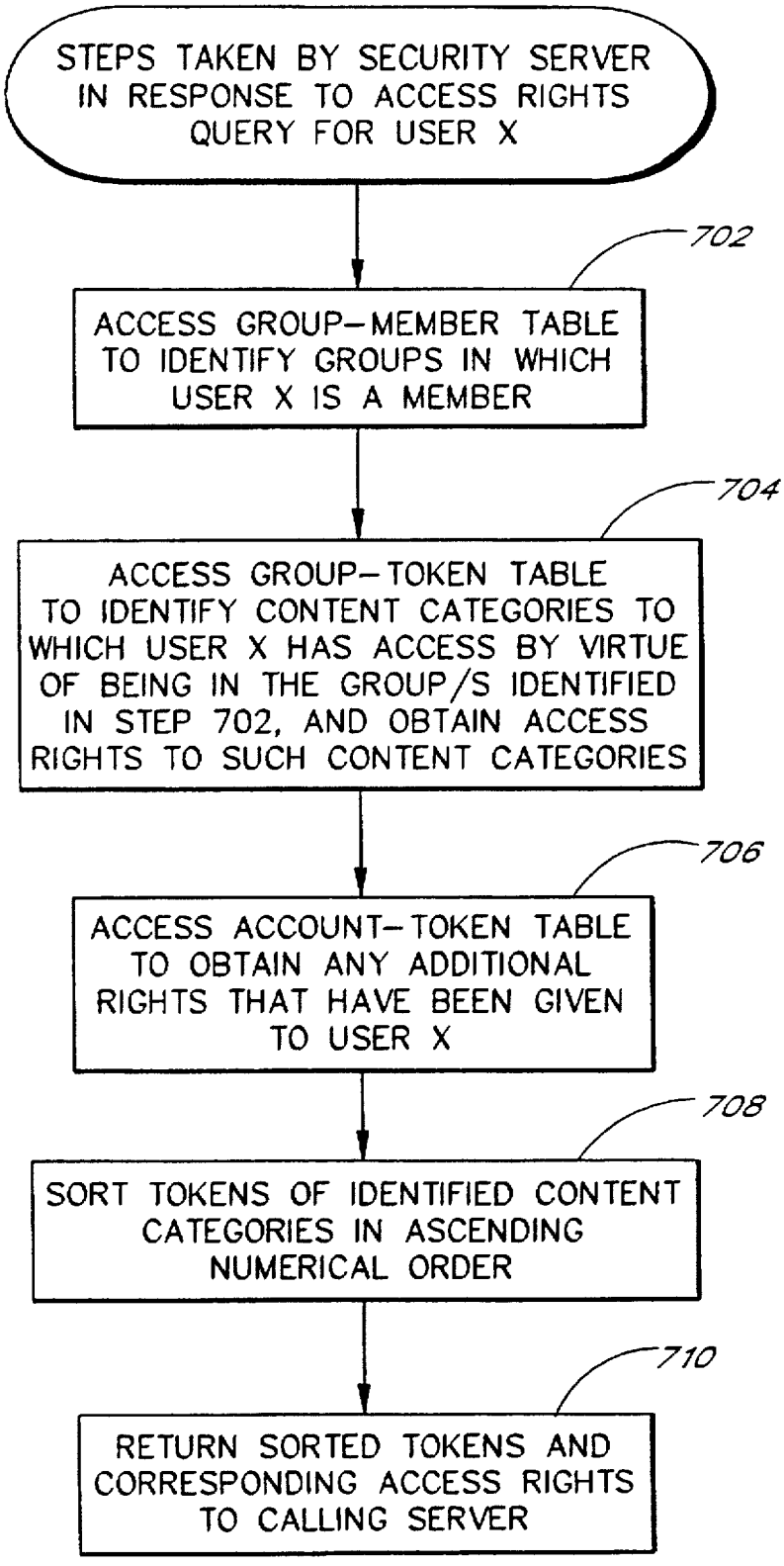

STEPS TAKEN BY SECURITY SERVER IN RESPONSE TO ACCESS RIGHTS QUERY FOR USER X
702
ACCESS GROUP–MEMBER TABLE TO IDENTIFY GROUPS IN WHICH USER X IS A MEMBER
704
ACCESS GROUP–TOKEN TABLE TO IDENTIFY CONTENT CATEGORIES TO WHICH USER X HAS ACCESS BY VIRTUE OF BEING IN THE GROUP/S IDENTIFIED IN STEP 702, AND OBTAIN ACCESS RIGHTS TO SUCH CONTENT CATEGORIES
706
ACCESS ACCOUNT–TOKEN TABLE TO OBTAIN ANY ADDITIONAL RIGHTS THAT HAVE BEEN GIVEN TO USER X
708
SORT TOKENS OF IDENTIFIED CONTENT CATEGORIES IN ASCENDING NUMERICAL ORDER
710
RETURN SORTED TOKENS AND CORRESPONDING ACCESS RIGHTS TO CALLING SERVER

FIG. 7

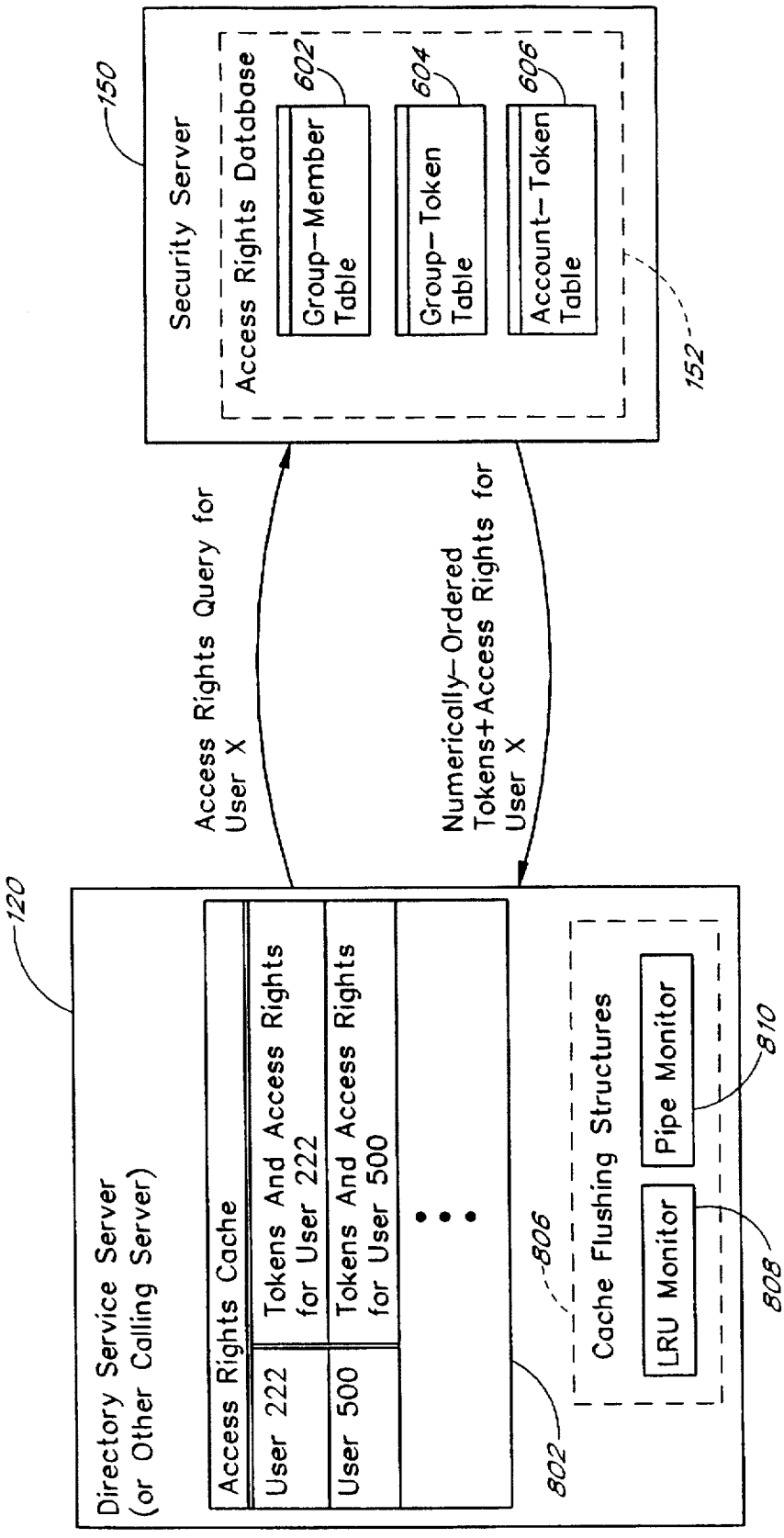

150
Security Server
Access Rights Database
602
Group–Member Table
604
Group–Token Table
606
Account–Token Table
152
Access Rights Query for User X
Numerically–Ordered Tokens+Access Rights for User X
120
Directory Service Server (or Other Calling Server)
Access Rights Cache
User 222
Tokens And Access Rights for User 222
User 500
Tokens And Access Rights for User 500
802
806
Cache Flushing Structures
LRU Monitor
Pipe Monitor
808
810

FIG. 8

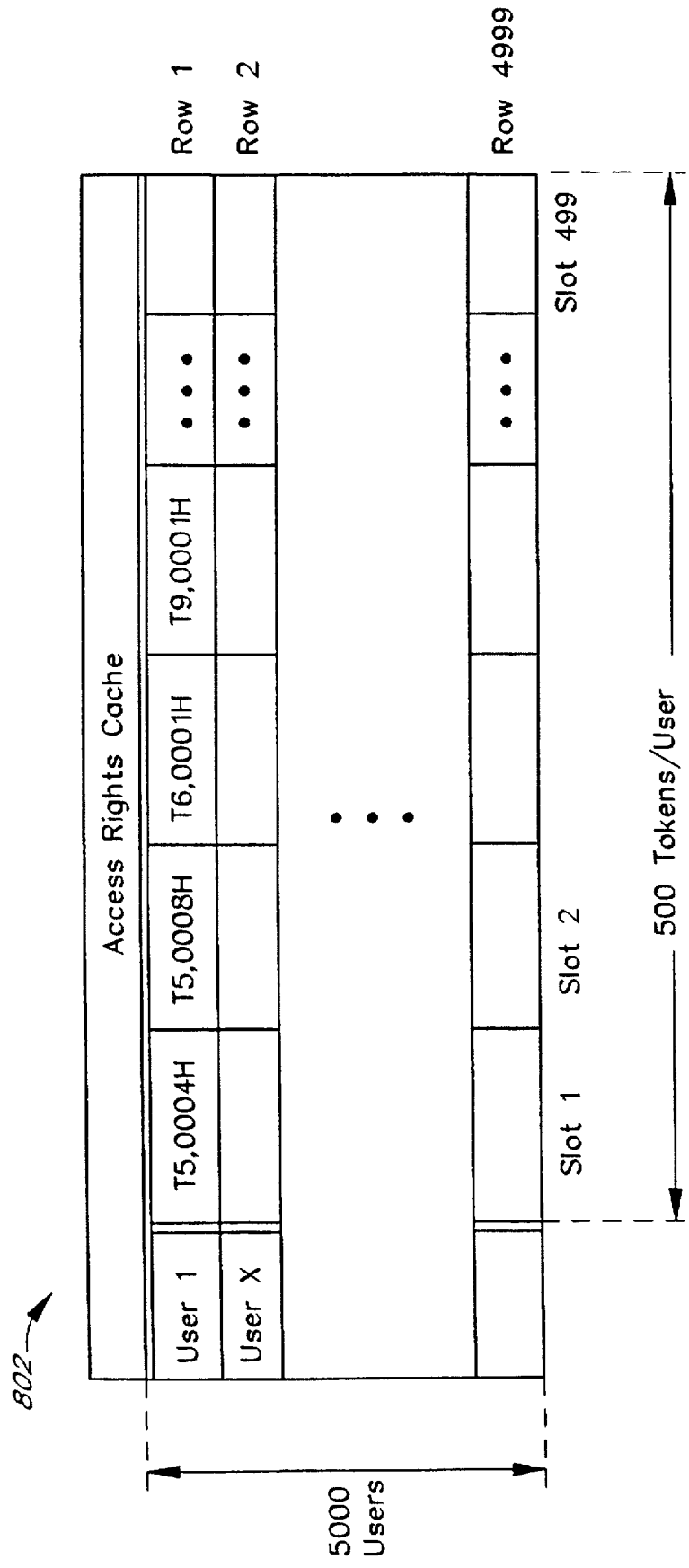

Row 1
Row 2
Row 4999
Access Rights Cache
T5,0004H
T5,0008H
T6,0001H
T9,0001H
User 1
User X
Slot 1
Slot 2
Slot 499
500 Tokens/User
5000 Users
802

FIG. 9

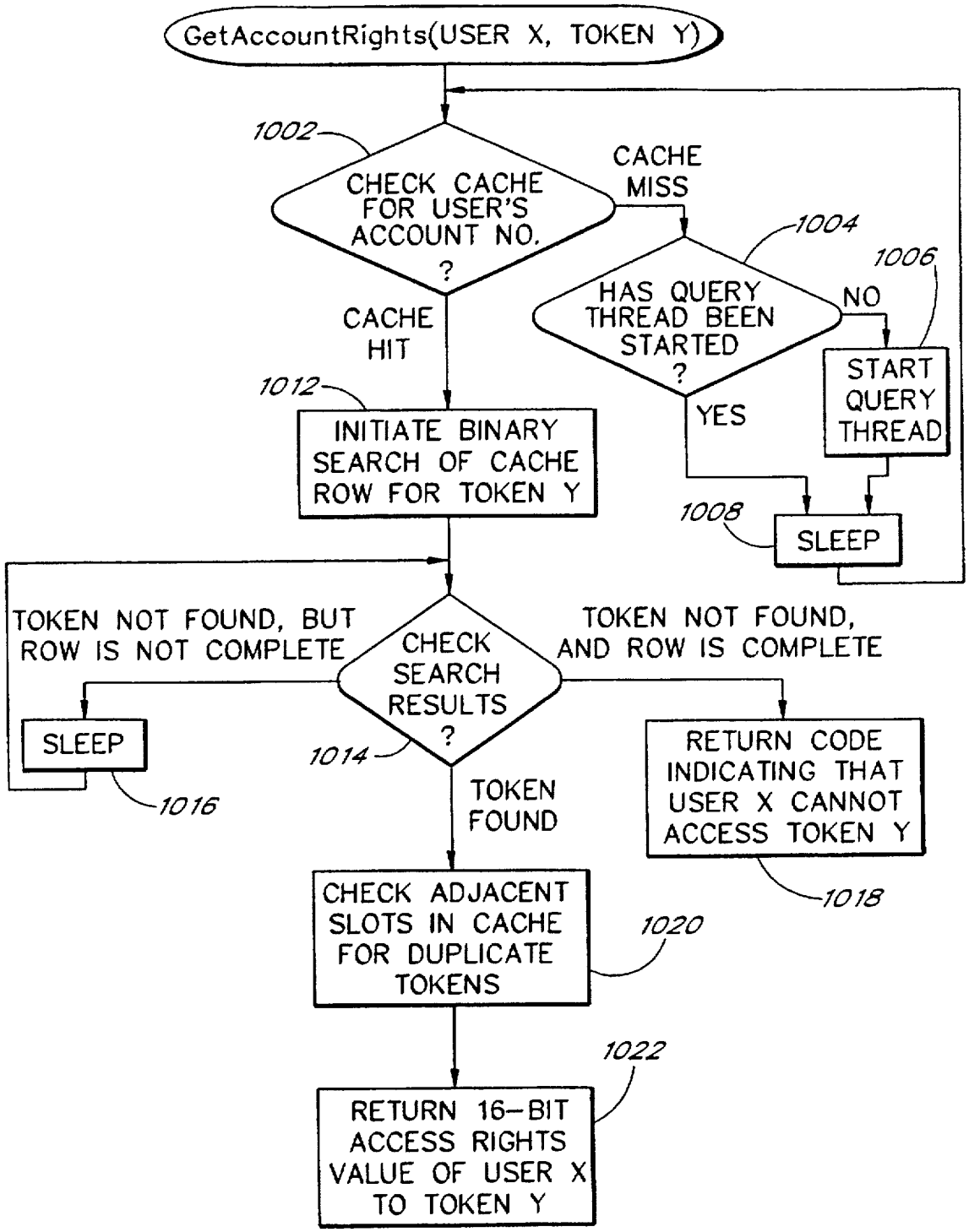

GetAccountRights(USER X, TOKEN Y)
1002
CHECK CACHE FOR USER'S ACCOUNT NO. ?
CACHE MISS
CACHE HIT
1004
HAS QUERY THREAD BEEN STARTED ?
NO
1006
START QUERY THREAD
YES
1008
SLEEP
1012
INITIATE BINARY SEARCH OF CACHE ROW FOR TOKEN Y
TOKEN NOT FOUND, BUT ROW IS NOT COMPLETE
CHECK SEARCH RESULTS ?
1014
TOKEN NOT FOUND, AND ROW IS COMPLETE
SLEEP
1016
TOKEN FOUND
RETURN CODE INDICATING THAT USER X CANNOT ACCESS TOKEN Y
1018
CHECK ADJACENT SLOTS IN CACHE FOR DUPLICATE TOKENS
1020
1022
RETURN 16-BIT ACCESS RIGHTS VALUE OF USER X TO TOKEN Y

FIG. 10

SYSTEM AND METHOD FOR CONTROLLING ACCESS TO DATA ENTITIES IN A COMPUTER NETWORK

FIELD OF THE INVENTION

The present invention relates to computer networks in which access rights to data entities vary from user-to-user. More particularly, the present invention relates to database systems for storing access rights information.

BACKGROUND

The present invention is directed generally to the problem of flexibly and efficiently controlling the access rights of a large number of users to a large number of objects or other data entities. The problem arises, for example, in the context of on-line services networks in which end users are given differing levels of access to different content entities. These content entities represent the services or "content" of the network, as seen by end users. The content entities may include, for example, bulletin board messages, mail messages, data files, folders, image files, sound files, multimedia files, executable files, on-line services, connections to other networks, etc. An on-line services network of this type is described in copending U.S. Application Ser. No. 08/472,807 having the title ARCHITECTURE FOR SCALABLE ON-LINE SERVICES NETWORK, filed Jun. 7, 1995 (Now U.S. Pat. No. 5,774,668).

The need to assign user-specific access rights to different content entities arises in a variety of situations. For example, it may be desirable to give some users access to certain "premium" services (such as specially-targeted investment newsletters), while limiting others users to some basic set of services. Further, it may be desirable to give certain users (such as system operators or administrators) the ability to modify, rename or delete certain content entities (such as bulletin board messages), while limiting other users to read-only access of such entities.

Various techniques are known in the art for controlling user accesses to objects and other data entities. One technique, which is commonly used in file systems, involves the storage of an access control list (ACL) for each data entity to which access is to be controlled. The ACL for a given data entity will typically be in the form of a list of the users that have access to the data entity, together with the access rights of each such user with respect to the data entity. Each time a user requests access to the entity, the data entity's ACL is searched to determine whether the requested access is authorized. Another technique involves the storage of a capabilities list for each user. The capabilities list for a given user will typically include a list of the objects to which the user has access, together with the operations that can be performed by the user on each listed object. Both the ACL technique and the capabilities list technique are described in Silberschatz and Galvin, *Operating System Concepts, Fourth Edition*, Addison-Wesley Publishing Company, 1994.

With the increasing popularity of on-line services networks, and with the increasing need for such networks to provide limited user access to the Internet, it has become increasingly important to be able to provide large numbers of users with controlled access to large numbers of content entities. In the network described in the above-referenced application, for example, it is contemplated that the number of subscribers may be in the millions, and that the number of content entities may be in the tens of thousands. To provide flexibility, it is also desirable to be able to individualize the access rights of users.

Although prior art access control techniques such as those summarized above are suitable in theory for flexibly controlling user access in large-scale on-line services networks, these techniques tend to produce prohibitively large quantities of access rights data. For example, in a network having millions of users, the access control list technique might produce access control lists that have millions of entries. These large quantities of access rights consume large amounts of memory, and often take unacceptably long periods of time to search.

A need thus exists in the art for a technique that is suitable for flexibly controlling the access of a large number of users to a large number of data entities. A need also exists to be able to flexibly and efficiently define new types of access operations as new on-line services and new content entities are created.

SUMMARY

In accordance with the present invention, there is provided a system and method for controlling user access to data entities in a computer network. The data entities are preferably in the form of content objects of an on-line services network, although the system and method can be used to control access to other types of data entities.

In a preferred implementation of an on-line services network in which the present invention is embodied, the content objects are stored on multiple application servers of the network, and represent the on-line services and service data that is accessible to users of the network. Examples of content objects include bulletin board system (BBS) messages and folders, Chat conferences, download-and-run files, and service applications which implement specific on-line services. Users access these content objects by connecting to different application servers and corresponding services in the course of a logon session.

Service applications running on the application servers implement various on-line services, such as Chat, Mail, BBS, FTM (File Transfer Manager) and Mediaview. One on-line service, referred to as the Directory Service, maintains a directory structure of the content objects that are accessible to users, with the content objects forming nodes of the tree-like directory structure. By sending properties of these nodes to a client application running on the computer of an end user, the Directory Service provides the user with a hierarchical, navigable view of the content of the network.

In accordance with the invention, different users of the network (including both subscribers and system administrators) are given different access rights with respect to different content objects, and can thus perform differing types of operations with respect to the content objects. For example, with respect to a given BBS folder, some users may be prevented from seeing or otherwise accessing the folder, some may be given read-only access to the contents of the folder, some may be given the capability to create new messages within the folder, and some may additionally be given the capability to delete and/or rename messages within the folder.

In accordance with one feature of the present invention, the access rights of the users of the network with respect to the various user-accessible content objects are specified by access rights data that is stored within an access rights database. The access rights database is implemented as a relational database on one or more security servers, which are connected to the application servers by a local area network. The access rights data is stored within the relational database in association with multiple content category

identifiers, or "tokens," which identify categories or groupings of content objects (such as "internal public data," "Internet public data," and "18-and-older only data") for security purposes. The various content categories are preferably defined by system administrators. The content categories, rather than the content objects, serve the basic content units to which user access rights may be specified. The use of content categories eliminates the need to store access rights data on a per-object basis, and thereby significantly reduces the quantity of access rights data that needs to be stored.

The access rights data is preferably stored within the relational database in further association with multiple user group identifiers, which identify user groups (such as "everyone," "allsysops," and "guests") that have been formed for the purpose of storing access rights data. By storing access rights data primarily on a per-user-group basis, rather than separately storing the access rights of each individual user, the use of user groups further reduces the quantity of access rights data that needs to be stored.

The use of content categories and user groups advantageously allows access rights to be specified for large numbers of users (typically millions) with respect to large numbers of content objects (typically thousands) with a high degree of granularity.

In accordance with another feature of the invention, the service applications running on the various application servers initiate user-specific queries of the access rights database to obtain access rights lists of specific users. With each user-specific access rights query, the security server that receives the query accesses the access rights database and generates an access rights list which fully specifies the access rights of the user. This access rights list is returned to the application server that generated the query, and is stored within an access rights cache of the application server. The service which initiated the query can then rapidly determine the of access rights of the user with respect to specific content objects (as described below) by accessing its locally-stored copy of the user's access rights list. Because a user may be connected simultaneously to multiple application servers of the on-line services network (when, for example, the user opens multiple services), the access rights list of a given user may be stored concurrently within the respective caches of multiple application servers.

In accordance another feature of the invention, the access rights list of each user includes pairs of tokens and corresponding access rights values. Each token in the list identifies a content category to which the user has at least some access rights. For example, a token of "5" in the list indicates that the user has access to all content objects which fall within content category **5**. Each access rights value in the list specifies the access rights of the user with respect to a corresponding content category. The access rights values are preferably in the form of privilege level masks which specify one or more general privilege levels (such as "viewer," "user," "host," "sysop," and "supersysop"). These general privilege levels are translated into specific sets of access capabilities by the on-line service applications. For example, the BBS service may give users with sysop-level privileges the capability to delete and rename BBS messages.

In accordance with another feature of the invention, when it becomes necessary for a service (running on an application server) to determine the access rights of a user with respect to a specific content object, the service initially reads the object's token, which is preferably stored as a property of a corresponding Directory Service node. This token specifies the content category to which the content object belongs. The service then generates an API (application program interface) call, which causes the application server to search its access rights cache for the user's access rights list, and if found, to search the access rights list for the token. If the user's access rights list is not found, the API initially generates a query of the access rights database (to fill the cache with the user's access rights list), and then begins to search the cache for the token. If the token is found, the API returns the corresponding access rights value to the service that generated the API call. If the token is not found, the API returns a code indicating that the user cannot access the content object.

In accordance with yet another feature of the present invention, the relational, access rights database includes three tables. The first table is a group-member table which specifies the user groups and the members (i.e., user accounts) of each user group. Each user of the network is a member of at least one user group, and may be a member of multiple groups. The second table is a group-token table which contains, for each user group, a group-based access rights list (in the form of a list of tokens and corresponding access rights values). Each group-based access rights list specifies the group-based rights which are provided to all members of the respective group. The third table is an account-token table, which specifies, on a single-user basis (for certain users), additional rights that are to be added to the group-based rights of the user. Each user-specific entry in the account-token table is preferably in the form of a single token plus a corresponding access rights value.

In addition to (or in place of) the account-token table, an exclusion table may optionally be implemented to specify access rights that are to be taken away from the accounts of specific users. As with the account-token table, each user-specific entry in the exclusion table is preferably in the form of a single token plus a corresponding access rights value. The exclusion table is useful, for example, for taking away certain privileges of users who misuse certain services.

Upon receiving a user-specific access rights query, the security server initially accesses the group-member table to identify all user groups of which the specified user is a member. The security server then accesses the group-token table to obtain the group-based access rights list of each user group of which the user is a member. The security server thereby identifies all of the rights the user has by virtue of being a member of one or more user groups. If the user is a member of multiple user groups, the multiple group-based access rights lists are combined so that the user is given all of the rights associated with all user groups of which the user is a member. The security server then accesses the account-token table to determine whether any additional (or "special") rights (in the form of tokens and corresponding access rights values) have been added to the account of the user. If one or more entries exist in the account-token table for the user, these entries are combined with the user's group-based rights to generate the user's access rights list. (For embodiments that include an exclusion table, if one or more entries exist for the user in the exclusion table, these entries are subtracted from the user's group-based rights.) The access rights list is then sorted such that the tokens of the list (and corresponding access rights values) are placed in numerically ascending order (to facilitate cache searches of the list), and the sorted list is transmitted to the application server that generated the query.

The system and method of the present invention advantageously enabled system administrators to flexibly control

user access to different "service areas" in order to achieve a variety of objectives. In accordance with a preferred mode of operation, when a new service area (preferably represented by one or more nodes of the directory structure) is created, a security token may be assigned to the new service area to provide separate security for the area. A particular user, who may be either a subscriber to the network or a system administrator, may then be given sysop-type privileges (via the above-mentioned account-token table) to the new service area. By making different users sysops with respect to different service areas, the responsibility of monitoring user-generated content is distributed among many different individuals. In accordance with another preferred mode of operation, content categories and user groups are formed so as to create many different communications forums (such as Chat conferences and BBS folders) for private correspondence among user-specified subgroups of users.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features of the invention will now be described with reference to the drawings of a preferred embodiment, which is intended to illustrate and not to limit the invention, and in which:

FIG. **1** is a high level diagram illustrating the general architecture of an on-line services network which provides access control in accordance with the present invention.

FIG. **2** illustrates how the content of the on-line services network of FIG. **1** is preferably arranged within a tree-like directory structure of content nodes.

FIG. **3**A illustrates an access control matrix which specifies, for each user and for each node of the directory structure of FIG. **2**, whether the user can access the node, and if so, what the level of access is. The notation "XXXX" in FIG. **3**A represents a 16-bit access rights value.

FIG. **3**B illustrates a preferred basic set of privilege levels, and illustrates one possible assignment of access rights bits to the privilege levels.

FIG. **4**A illustrates how the access control matrix of FIG. **3**A is preferably compressed horizontally by the assignment of content nodes to content categories, with each content category identified by a numerical security token.

FIG. **4**B is a token definition table which illustrates a preferred basic set of security tokens (tokens **1–4**), and which illustrates examples of tokens (tokens **100** and **101**) which may be added to accommodate specific data types.

FIG. **5**A illustrates how the access control matrix of FIG. **3**A is compressed vertically by the assignment of users to user groups.

FIGS. **5**B is a group definition table which shows a preferred basic set of user groups, and which illustrates one possible assignment of group IDs to user groups.

FIG. **6** illustrates a preferred relational database which is used to store access rights data in accordance with the present invention. Numerical values in FIG. **6** are examples of possible table entries.

FIG. **7** illustrates a sequence of steps taken by one of the security servers of FIG. **1** when a database query is made for the access rights of a specific user.

FIG. **8** illustrates the preferred process by which one application server queries a security server for the access rights of a specific user and then caches the access rights data returned by the security server. Also shown in FIG. **8** are the basic structures used for flushing user-specific rows of the cache.

FIG. **9** illustrates a preferred arrangement of the cache of FIG. **8**. Numerical values in FIG. **9** correspond to the example table entries of FIG. **6**.

FIG. **10** illustrates a sequence of steps taken by an application server to determine the access rights of a specific user ("user X") with respect to a specific token ("token Y").

Reference numbers in the drawings have three or more digits; the two least significant digits are reference numbers within the drawing, and the more significant digits indicate the figure in which the item first appears. For example, reference number **602** refers to an item which is first shown in FIG. **6**. Like reference numbers indicate like or functionally similar components.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

Described herein is a system and method for controlling the access rights of users of an on-line services network to content entities such as bulletin board messages, message folders, chat conferences, service applications, download-and-run files and data files. As will be recognized by those skilled in the art, the system and method of the present invention are generally independent of the specific type or types of data entities to which access is being controlled. For example, the data entities could be low-level software and/or hardware resources such as threads, semaphores, memory segments and CPUs. It will further be recognized that the system and method of the present invention could be employed in any of variety of alternative networking environments, including file systems and operating systems.

For convenience, the description of the preferred embodiment is broken up into the following 12 sections:

1. ARCHITECTURAL OVERVIEW (FIG. **1**);
2. OVERVIEW OF CHAT AND BBS SERVICES;
3. OVERVIEW OF DIRECTORY SERVICE AND SECURITY (FIG. **2**);
4. ACCESS RIGHTS (FIGS. **3**A AND **3**B);
5. COMPRESSION BY GROUPING OF OBJECTS (FIGS. **4**A AND **4**B);
6. COMPRESSION BY GROUPING OF USERS (FIGURES **5**A AND **5**B);
7. ACCESS RIGHTS DATABASE (FIG. **6**);
8. QUERIES OF ACCESS RIGHTS DATABASE (FIGS. **7** AND **8**);
9. ACCESS RIGHTS CACHE (FIG. **9**);
10. GetAccountRights METHOD (FIG. **10**);
11. ASSIGNMENT OF TOKENS AND FORMATION OF USER GROUPS; and
12. OTHER EMBODIMENTS

The first of these sections provides an overview of the on-line services network in which the present invention is employed. The architecture of this network is further described in the above-referenced, commonly assigned application having the title "ARCHITECTURE FOR SCALABLE ON-LINE SERVICES NETWORK" (U.S. Ser. No. 08/472,807), which is incorporated herein in by reference.

1. Architectural Overview (FIG. **1**)

FIG. **1** is a high level diagram illustrating the general architecture of an on-line services network **100** which provides access control in accordance with the present invention. Multiple client microcomputers **102** are connected to a host data center **104** by a wide area network (WAN) **106**. The wide area network **106** includes WAN lines **108** which

are provided by one or more telecommunications providers, and which allow end users (i.e., users of the microcomputers **102**) over a wide geographic area to access the host data center **104**. The WAN lines **108** may include, for example, X.25 lines, TCP/IP lines, and ISDN (Integrated Service Digital Network) lines. The host data center **104** provides a variety of information-related and communications-related on-line services to end users.

The host data center **104** comprises a plurality of application servers **120** connected to one or more high speed local area networks (LAN) **122**. The application servers **120** are preferably Pentium-class (or better) microcomputers which are scalable to at least four central processing units (CPUS), and which run the Microsoft Windows NT operating system available from Microsoft Corporation. Each application server **120** typically has at least 128 MB of random-access memory (RAM) and at least 4 GB of disk space.

The application servers **120** are arranged into service groups (also referred to as "AppGroups") that correspond to specific on-line services. Each service group runs a particular service and provides access to a corresponding data set. Three example service groups are shown in FIG. **1**: a CHAT service group **130**, a bulletin board system (BBS) service group **132**, and a DirSrv service group **134**. Additional service groups (not shown) are provided to implement other on-line services, including Mediaview (a service which provides multimedia titles to end users), Mail (an email service), FTM (a service for uploading and downloading files) and Component Manager (a service which allows users to update client software when new releases become available). Other on-line services may include, for example, an interactive games service, a file transfer service, a weather service, and a World Wide Web browsing service. A service group can have as few as one application server **120**. System administrators can adjust the number of application servers **120** in a given service group to accommodate the current usage level of the corresponding service.

Also connected to the LAN **122** are multiple Gateway microcomputers **140** (hereinafter "Gateways") which link incoming calls from end users to the application servers **120**. The Gateways are preferably Pentium-class microcomputers which are scalable to at least four central processing units (CPUs), and which run the Microsoft Windows NT operating system. Each Gateway **140** typically has at least 64 MB of RAM and at least 2 GB of disk space, and is capable of supporting approximately 1000 simultaneous user connections.

Also connected to the LAN **122** are multiple security servers **150**. The security servers **150** are preferably Pentium-class microcomputers which are scalable to at least four central processing units (CPUs), and which run the Microsoft Windows NT operating system. Each security server **150** maintains a relational database **152** (i.e., a database in which the contents are organized as a set of two or more interrelated tables) which contains the access rights data for all users of the network **100**. In the preferred embodiment, the security servers **150** are replicated, meaning that they store and provide access to the same access rights data. In other embodiments, the access rights data may be partitioned across the security servers **150**.

Each security server **150** runs Structured Query Language (SQL) code to provide access to its respective access rights database **152**. SQL is a programming language standardized by the International Standards Organization (ISO) for defining, updating and querying relational databases. A query to the access rights database **152** can emanate either from one of the application servers **120** (when, for example, a user attempts to access a content object which is stored by the application server **120**), or by one of the Gateways **140** (when a user attempts to open an on-line service). In accordance with one feature of the present invention, each machine **120**, **140** which generates queries of the access rights database **152** implements an access rights cache for locally storing user-specific access rights information obtained from the database **152**. In other embodiments, all access rights queries may be generated by a single group or type of machine (e.g., the Gateways **140**, or a group of logon servers), and these machines may be configured to pass the user-specific access rights information read from the database **152** to the various application servers **120** to which the user connects.

Various other types of servers and other microcomputers are connected to the LAN **122** but are not shown in FIG. **1**. For example, billing and logon servers are provided to record billable events and to handle user logon, respectively. Further, Arbiter microcomputers are provided to perform transaction replication services for certain service groups, allowing the application servers of such service groups to store identical copies of the same service content data.

It is envisioned that the host data center **104** may have on the order of one hundred Gateways **140**, and between several hundred and several thousand application servers **120**. A host data center of this type will be able to handle millions of subscribers and tens of thousands of simultaneous user logon sessions. Advantageously, the processing capacity of the host data center **104** can easily be increased (to support new services, and to support increases in the number of subscribers) by connecting additional Gateways **140** and application servers **120** to the LAN **122**, and by adding additional local area networks. Further, additional host data centers **104** can be provided at different geographical locations to accommodate a wide geographic distribution of subscribers.

"Users" of the on-line services network **100** include both "end users" (typically subscribers) who log onto the system from client microcomputers **102** via the WAN **106**, and "internal" users (typically system administrators) who access the system from computers that are connected directly to the LAN **122**. Each user of the network, whether an end user or an internal user, is identified by a unique 32-bit account number. As described below, different users have different access privileges with respect to various data entities on the network.

The on-line services offered to end-users of the network **100** are in the form of client-server applications programs (or "service applications"). Each service application includes a server portion that runs on one or more of the application servers **120**, and at least one corresponding client portion (also referred to as a "client application") that runs on a microcomputer **102** of an end user. In the presently preferred embodiment, the client applications are in the form of Microsoft Windows 95 components (including dynamic link libraries, other executables, and data files), and the server portions are implemented primarily as dynamic link libraries running under the Microsoft Windows NT Operating System.

With reference to FIG. **1**, the server portions of the various on-line services are implemented on the application servers **120** of the respective service groups **130**, **132**, **134**. Each application server **120** of a given service group separately runs the same server application. For example, each application server **120** of the Chat service group **130** runs CHAT.DLL, which is a dynamic link library that implements the server portion of the Chat service. Similarly, each

application server **120** of the BBS service group **132** runs a BBS dynamic link library, and each application server **120** of the DirSrv service group **134** runs a DirSrv dynamic link library. Although each application server **120** is shown in FIG. **1** as being allocated to a single service group, a single application server can simultaneously run multiple service applications, and thus be allocated to multiple service groups. For example, a single application server **120** could run both the Chat and BBS dynamic link libraries and thus be allocated to both the Chat and BBS service groups **130**, **132**.

During a typical logon session, a client microcomputer **102** will maintain a communications link with a single Gateway **140**, but may access multiple on-line services (and thus communicate with multiple application servers **120**). To initially access a service, an "open" request is generated on the client microcomputer **102** and sent to the Gateway **140** that is handling the logon session. The Gateway **140** then selects a single application server **120** (of the appropriate service group) to handle the service session, and opens a pipe (or other type of connection) over the LAN **122** to the selected application server **120**.

Throughout the service session, the Gateway **140** routes messages between the client microcomputer **102** and the application server **120** as the client and server portions of the service application communicate. The Gateway **140** also performs protocol translation between the protocol of the WAN **106** and the protocol of the LAN **122**. To terminate the service session, a "close" request is generated at the client microcomputer **102** and sent to the Gateway **140**, and the Gateway **140** closes the pipe to the application server **120** that is handling the service session.

The architecture advantageously supports multiple simultaneous service sessions per user. Thus, a user may be connected to multiple applications servers (via the Gateway **140** handling the logon session) simultaneously.

2. Overview of Chat and BBS Services

Two specific on-line services, Chat and BBS, will now be briefly described. This description will illustrate some of the specific types of content entities (referred to herein as "content objects," or simply "objects") which may be accessed by users, and will also illustrate some of the different types of access rights users may be given with respect to such content objects.

The Chat service is an interactive communications service which allows users to have real time conversations with other users on specific topics. Chat conversations or "conferences" are organized as "Chat rooms" which may be entered or exited by end users to join or leave the corresponding conferences. For example, an end user may enter a "sports" Chat room to join an interactive conversation on sports-related topics. Participants in a Chat conference can type in textual messages which will be displayed on the monitors of other participants. Voice and/or video capabilities may additionally be provided.

The BBS service allows users to post and/or review messages. Users can thereby ask and answer questions, or otherwise conduct non-real-time conversations with other users. Although shown as a single BBS service group **132** in FIG. **1**, multiple BBS service groups may be formed, with each corresponding, for example, to a particular topical area. In the preferred implementation, replicated copies of all BBS content (e.g., BBS messages and folders) are stored on each application server **120** of the BBS service group **132**. This allows the BBS application servers **120** to independently process message read requests from end users. Replication of BBS content is accomplished using the Arbiter transaction replication service. A preferred embodiment of the Arbiter service is described in commonly assigned U.S. application Ser. No. 08/485,493, filed Jun. 7, 1995, having the title TRANSACTION REPLICATION SYSTEM AND METHOD FOR SUPPORTING REPLICATED TRANSACTION-BASED SERVICES.

With reference to FIG. **1**, one of the application servers **120** of the BBS service group **132** is preferably configured as an Internet feed server **120**. The BBS Internet feed server **120** reads Internet newsgroup messages and posts these messages (by submitting update transactions to the Arbiter service) within the BBS service group **132**, thereby providing users with access to such newsgroup messages. The BBS Internet feed server **120** is also used to post messages to the Internet.

Chat rooms and BBS messages are two types of content objects that may be accessed by users. BBS folders (objects which contain BBS messages and/or other BBS folders) are another type of content object that may be accessed.

The ability to access a given content object, and the access rights of the user with respect to that object, may vary from user to user. Using a Chat room object as an example, some users may be "participants" who can participate in the conference, while other users may be "viewers" who can only view the text of the conversation. One user may further be designated as the "host" of the conversation. The host normally has the responsibility of moderating the conversation, and has the ability to modify the access rights of members of the conversation. For example, if a user fails to comply with the rules of the Chat conference, the host can set that user's privilege level to "viewer," preventing the user from further participating in the conversation. Access rights of users are preferably controlled (typically by system operators or administrators) by updating entries in the access rights database **152**, as described in detail in the following sections.

As with Chat objects, the access rights of users with respect to different BBS objects (e.g., BBS folders and messages) may vary from user to user. For example, certain BBS folders may be designated as "public," meaning that they can generally be accessed by all users, while other BBS folders may be designated as "private," meaning that access to such folders is restricted to some subgroup of users. A private folder may be used, for example, to permit private personal correspondence between a user-specified group of family and friends.

The specific types of operations allowed with respect to a BBS object may vary from user to user. For example, some users may have read-only access within a BBS folder, in which case they will not be able to reply to an existing BBS message in that folder and will not be able add a new message to the folder. Other users may be able to add new BBS messages to the folder and/or reply to existing messages, but not delete existing messages.

Other users, generally referred to as "sysops" (system operators), may be given the ability to delete existing messages from the folder. Different end users can be designated by the on-line services network provider (i.e., the owner or operator of the host data center **104**) as the sysop for a particular folder or group of folders. Thus, for example, a particular end user may be placed in charge of a football BBS folder, while another end user may be placed in charge of a baseball BBS folder. This advantageously allows the on-line services network provider to distribute the responsibility of monitoring BBS content among a large number of end users.

Users at the system administrator level may be given the additional capability of creating new BBS folders, deleting

existing BBS folders, and/or changing the access rights of users with respect to BBS folders.

The foregoing examples illustrate some of the specific types of access privilege levels which may be assigned to users with respect to certain object types, and illustrate some of the reasons for assigning different levels of access rights. As additionally illustrated by the foregoing, it is often desirable to define different (and often unique) types of accesses for different on-line services and object types. For example, for the Chat service, it is desirable to have the above-described viewer, participant and moderator type access privileges, even though the operations corresponding to these privileges are generally unique to the Chat service.

It will also be recognized that as new on-line services and new object types are added to the network **100**, it may be necessary or desirable to define new types of access operations. To facilitate the addition of new on-line services and object types, the network **100** provides for a specified set of privilege levels (such as "viewer," "observer," "user," "host," "sysop," and "sysop manager") which can be assigned to users, and it is left to the on-line services themselves (i.e., to the authors of the service applications) to define the specific access capabilities that go along with each user privilege level. For example, for a user that has been assigned the general privilege level of "user," the Chat service may give the user "participant" level access to all public Chat rooms, while the BBS service may allow the user to read, generate and reply to BBS messages within all public BBS folders. This feature of the present invention is further described below under the heading ACCESS RIGHTS.

3. Overview of Directory Service and Security (FIG. **2**)

The following is an overview of the Directory Service, which is an on-line service that allows users to explore the content (i.e., the on-line services and associated data entities) of the network **100**. The Directory Service is described in detail in a concurrently filed U.S. application having the title DIRECTORY SERVICE FOR A COMPUTER NETWORK, which is incorporated herein by reference. Included in this overview is a brief description of how the Directory Service and other services determine the access rights of users with respect to specific content objects.

The Directory Service provides users with a hierarchical view of the various content objects available on the network **100**. As further described below, the content objects are arranged within hierarchical directory structures **202**, **204** (FIG. **2**) that are maintained by the Directory Service, with the content of the content objects represented as nodes of these structures **202**, **204**.

The content of the network **100** is displayed to the end user via a network shell program which runs on the client microcomputers **102** of end users. The network shell is the primary client of the Directory Service. A preferred implementation of the network shell is described in a commonly-assigned U.S. application having the title ON-LINE NETWORK ACCESS SYSTEM, filed Jul. 17, 1995. In the preferred embodiment, the network shell is an integral part of the Microsoft Windows 95 Explorer program (hereinafter "the Explorer") which is described in *Inside Windows*95, Microsoft Press, 1994.

A graphical user interface of the Explorer displays the content objects as a logical extension of the user's hard drive, with each object shown as an icon and/or a textual name. Using the Explorer, users can browse the content of the network **100**, and can access the various content objects (to, for example, enter a specific on-line service). To access a content object, the user double clicks on the icon for that object. As further described below, the Directory Service only "shows" those content objects to which the particular user has access. Thus, the user is provided with a filtered view of the actual content of the network **100**.

With reference to FIG. **1**, the Directory Service (abbreviated as "DS" in FIG. **1**) includes two separate services, the DirSrv service (implemented on the DirSrv service group **134**) and the BBS service (implemented on the BBS service group **132**). The DirSrv service is the "root" of the Directory Service, and provides users with a hierarchical, navigable view of all non-BBS content objects. These non-BBS content objects are arranged within the DirSrv directory structure **202**. The BBS service acts as its own directory service provider, and provides users with a navigable, hierarchical view of all BBS content objects. The BBS content objects are arranged within the BBS directory structure **204**. A seamless interface between the DirSrv and BBS services allows users to transparently traverse between the two directory structures **202**, **204**, so that the Directory Service appears as a single service to end users, and so that the two directory structures **202**, **204** appear as a single tree-like directory structure.

The DirSrv and BBS services are both "directory service providers," meaning that they act as the Directory Service with respect to corresponding portions of the network content. Additional directory service providers can be added to the Directory Service as the content of the network **100** grows. For example, an investment service that provides data on stocks and mutual funds could be added which acts as a directory service provider with respect to its own content.

FIG. **2** illustrates the general organization of the content objects within the directory structures **202** and **204**, as maintained by the Directory Service. Each content object is represented as a corresponding node of one of the directory structures **202**, **204**. The first directory structure **202** exists within the DirSrv namespace **212**, and represents the content that is accessible through the DirSrv service. The second hierarchical structure **204** exists within the BBS namespace **214**, and represents the content that is accessible through the BBS service. Each structure **202**, **204** may have thousands of nodes, and could thus represent thousands of content objects. The nodes can generally be thought of as "service areas" that can be entered by users. Links between nodes represent paths that can be taken by users in traversing the hierarchical structures **202**, **204** from one service area to another. The specific nodes and links shown in FIG. **2** are provided to show the general manner in which nodes are arranged, and do not represent an existing directory structure.

The hierarchical directory structures **202**, **204** are preferably in the form of directed acyclic graphs. As is well known in the art of file systems, an acyclic graph structure is more flexible than a tree structure, since an acyclic graph allows a node to have multiple parent nodes. (A "parent" of a given node is any node that is both (1) directly connected to the given node, and (2) at a higher level in the hierarchy than the given node. Similarly, a "child" is any node that is both (1) directly connected to the given node, and (2) at a lower level than the given node.) This characteristic of the directory structures **202** and **204** is illustrated by nodes **10** and **17**, each of which has two parent nodes. To simplify the following description, the term "Directory Service structure" will be used to refer collectively to the DirSrv and BBS directory structures **202** and **204**.

There are three different types of nodes within the Directory Service structure: leaves, folders and junction points. A

set of flags stored in association with each node identifies the node as one of these three types. Leaves (or "leaf nodes") are nodes that both (1) cannot have children and (2) do not serve as junction points. The leaf nodes in FIG. **2** are nodes **7–11**, **16** and **17** (assuming that these nodes cannot have children). Leaves normally represent the actual services within network **100**. Examples of leaves include Chat rooms, BBS messages, Mediaview titles and download-and-run files. When the user clicks on a leaf node (by double clicking on the corresponding icon from a window of the Explorer client), the corresponding service-related action is taken. For example, if the user double clicks on a Chat room icon, the Chat service is opened and the user is added to the corresponding Chat conference. When the user double clicks on a leaf node for a download-and-run file, the file is downloaded to the user's computer **102** for execution.

Folders are nodes that both (1) can have children and (2) do not serve as junction points. The folder nodes in FIG. **2** are nodes **0–6** and **13–15**. Folder nodes normally represent collections of other content objects, and are used to organize the content of the network. For example, a folder node may correspond to a BBS folder on a particular topic, or may represent a collection of BBS folders and Chat rooms on a related topic. Folder nodes are also used to generally arrange content objects according to language. For example, node **1** may be an english folder containing content objects that are primarily in english, and node **2** may be a spanish folder containing content objects that are primarily in spanish. Folder nodes are generally analogous to the directories of a file system.

The third type of node is a junction point. Junction point nodes serve as proxies for nodes in other Directory Service namespaces, and are used to allow the user to seamlessly traverse between namespaces. The only junction point shown in Figure is node **12**, which serves as a proxy for BBS folder node **14** (referred to as the "target node"). When, for example, the user double clicks on node **12**, the Explorer launches a BBS navigator and shows the user the children of node **14**.

The Dirsrv and BBS services store their respective nodes as lists of node properties, as illustratively shown for node **8** in FIG. **2**. The DirSrv and BBS service also keep track of the locations of the nodes within their respective directory structures **202**, **204**. As pictorially illustrated in FIG. **1**, the full DirSrv directory structure **202** (i.e., the nodes within the DirSrv namespace **212** plus the arrangement of the nodes within the directed acyclic graph) is stored on each of the application servers **120** of the DirSrv service group **134**. Similarly, the full BBS directory structure **204** is stored on each of the application servers **120** of the BBS service group **132**. Depending upon the object type, certain of the node properties stored by the Directory Service may be service-specific. For example, BBS message nodes preferably include a BBS-specific "attachments flag" which indicates whether a file attachment is included with the message. Other properties are general in nature, and are shared by most or all of the Directory Service nodes. The following is a brief description of some of these general properties.

Name. This is a human readable name which may be displayed by the Explorer along with the corresponding icon. For example, a folder node could have the name "Health & Fitness," and could have children folder nodes with names of "Health & Fitness Chat" and "Heath & Fitness BBS." (For junction point nodes, the name of the target node is used).

Directory Entry ID (DEID). This is an 8-byte number which uniquely identifies a node within its respective Directory Service namespace. Every node has a DEID.

Application ID (APPID). This is a 4-byte number which is stored as a property of every node. For leaf nodes, this number identifies the service application associated with the node, and is used by the Explorer to launch the service application when the user double-clicks on the node. For non-leaf nodes, the APPID indicates the namespace (DirSrv or BBS) in which the node resides.

Service Group ID. (Also referred to as the data set ID.) This is a 2-byte number which identifies the service group (**132** or **134**) of the Directory Service provider.

Icon ID. This is an identifier of the icon which is to be displayed by the Explorer as a representation of the node. Icon bitmaps are stored by the Directory Service, and are sent over the network upon request by the Explorer.

Flags. The flags indicate whether the node is a folder, leaf, or junction point.

Security Token. This is a 4-byte value which identifies a content category to which the node has been assigned for security (i.e., access rights) purposes. When a user attempts to access a node, the node's security token and the user's 32-bit account number are used to determine the user's access rights with respect to the node. (For junction point nodes, the security token of the target node is used). Security tokens are described in detail below under the heading COMPRESSION BY GROUPING OF OBJECTS.

Although the terms "node" and "content object" will be used somewhat interchangeably throughout the following description, it should be understood that each node is simply a list of content object properties stored by the Directory Service. In the case of a leaf node, this list of properties will typically correspond to a content object which is stored on some other application server **120**. For a Chat room object which resides on a Chat server **120**, for example, the corresponding node will be a list of the properties for the Chat room, and will be stored on each of the DirSrv servers **120**. For folder nodes which simply represent groupings of other nodes, the folder node and folder content object are essentially the same entity.

Nodes of the Directory Service structure are preferably added, deleted and modified using "Sysop Tools," which is a client application of the Directory Service. As will be appreciated by those skilled in the art, various conventional editing tools can be used for this purpose. To create a node using Sysop Tools, the user must specify at least the DEID, APPID and the service group ID of the node. The Sysop Tools client is further described in a commonly-assigned, concurrently filed U.S. application having the title SYSTEM AND METHOD FOR EDITING CONTENT IN AN ON-LINE NETWORK.

The Directory Service operates generally as follows. In response to requests from the Explorer, the Directory Service sends node properties over the WAN **106** to the client microcomputer **102**, allowing the Explorer to reconstruct user-selected portions of the Directory Service structure on the user's screen, and/or allowing the Explorer to display user-specified object properties (such as the number of users in a Chat room) to the end user. To avoid unnecessary transfers of information over the WAN **106**, the Directory Service only returns those properties that are specifically requested by the Explorer. When the user double clicks on a folder node, the Explorer uses a GetChildren API (application program interface) to generate a request to the Directory Service for the children of the folder node, specifying as parameters of the API the DEID of the folder node plus the names of the specific properties needed to display

the children of the folder node. When the user double clicks on a leaf node, the Explorer initiates a service session with the corresponding service, using the leaf node's APPID to identify the appropriate service application.

Before "showing" a node to the end user (by returning the requested properties of the node to the Explorer), the Directory Service uses a GetAccountRights API to determine the access rights of the user with respect to the node (or equivalently, with respect to the corresponding content object), and to thereby determine whether the user is authorized to access the node. This access rights information is stored within the access rights database **152** on each security server **150**. If the user is not authorized to access the node, the Directory Service does not return the properties of the node, and the node is not displayed to the user. By way of example, suppose that a user double clicks on the icon corresponding to node **6** in FIG. **2**. This will cause the Explorer to send a GetChildren request to the Directory Service. As parameters of the GetChildren request, the Explorer specifies the DEID of node **6**, and specifies the properties (typically the name, DEID, APPID, flags and icon ID) to be returned for each child node. If, for example, the user is authorized to access node **7**, but is not authorized to access node **8**, the Directory Service will return only the properties of node **7**. Thus, node **8** will not appear in the Explorer window on the user's screen.

This feature of the invention advantageously allows certain nodes and content objects to be completely hidden from certain classes of users. For example, this feature may be used to hide from the view of regular users a BBS folder (and its contents) that has been created for private correspondence between members of a family, so that the only users who can see the folder (via the Explorer or other client application) are the designated family members. Because only those authorized to access each node can see the node, a high degree of security is provided against unauthorized accesses.

To determine the user's access rights with respect to the node, the Directory Service initially reads the 32-bit security token associated with the node (which, as described above, is stored as a node property). The Directory Service then generates a GetAccountRights call, specifying as parameters of the call the node's security token and the user's 32-bit account number. The GetAccountRights API returns either a 16-bit access rights value which indicates the user's access rights with respect to the node, or else returns a code indicating that the user is not authorized to access the node. The GetAccountRights API includes code which generates queries to the access rights databases **152** to obtain user-specific access rights lists, and also includes code which implements an access rights cache for locally storing these user-specific lists. The GetAccountRights API and a preferred implementation of the access rights cache are described in detail in sections **8–10** below.

In the preferred implementation of the network **100**, various forms of "direct navigation" are possible, wherein the user can access content objects without initially placing a Directory Service call. Using a "shortcuts" feature, for example, a user can create an icon that allows the user to subsequently return to a service area (such as a Chat room) without navigating the Directory Service structure. (The shortcuts feature is described in the above-referenced application of the title ON-LINE NETWORK ACCESS SYSTEM.) The Directory Service thus cannot be relied upon for ensuring the security of all content objects.

To ensure that the access rights of users are checked when direct navigation techniques are used, various other entities of the network **100** (in addition to the Directory Service) are preferably configured to call the GetAccountRights API. For example, the Chat service calls GetAccountRights to determine the rights of users with respect to Chat rooms, the Mail service calls GetAccountRights to determine whether users are authorized to send mail to specific distribution lists, and the FTM service calls GetAccountRights before downloading a file requested by a user. To provide an extra "layer" of protection, the Gateways **140** are preferably configured to call GetAccountRights whenever a user attempts to open a pipe to a service (as described below).

Although the architecture of the preferred embodiment allows a wide variety of different services and machines to generate queries of the access rights database **152**, it will be recognized that various alternatives are possible. For example, the network **100** may be configured such that the Directory Service is the only entity that generates queries of the access rights database **152**, and all access requests may then be routed through the Directory Service. Alternatively, the Gateways **140** or logon servers (not shown) could be configured to generate a query of the access rights database **152** when a user initially logs onto the network, and the user-specific access rights list obtained from this query may then be forwarded to each application server **120** to which the user connects. Both of these alternative approaches reduce the frequency of queries of the access rights database **152**.

4. Access Rights (FIGS. 3A and 3B)

FIG. **3A** illustrates an access control matrix **300** which represents the access rights of users of the on-line services network **100**. The information contained within the access control matrix **300** is stored in the access rights database **152** in a highly compressed form. Accordingly, the access control matrix **300** represents the information stored within the access rights database **152**, but does not represent the actual organization of this information within the database. The preferred methods used for compressing the access control matrix **300**, and the preferred implementation of the database **152**, are described in the following sections. As described below, the access control matrix **300** (and thus the access rights database **152**) specifies, for each user of the network, both (1) the content nodes that can be seen by the user via the Directory Service, and (2) the access operations that can be performed by the user with respect to each content node.

Each row of the access control matrix **300** corresponds to a respective user of the network **100**. These users include various levels of subscribers and system administrators. The number of users will typically be in the millions. Thus, the access control matrix **300** will typically have millions of rows. Each column of the access control matrix **300** corresponds to a respective node of the Directory Service structure of FIG. **2**. The total number of nodes in the Directory Service structure will typically be in the tens of thousands.

Each entry in the access control matrix **300** is in the form of a 16-bit access rights value (represented by the symbol "XXXX" in the figures), and specifies the access rights of a given user at a given node (or equivalently, specifies the rights of a given user with respect to a given content object). For example, the entry for user **1** at node **1** specifies the access rights user **1** has with respect to the content object corresponding to node **1** of the Directory Service structure. An entry of 0000H (in which "H" indicates the number is in hexadecimal) in the access control matrix **300** specifies that the user has no rights at the node, or equivalently, that the user cannot access the corresponding content object. The Directory Service will not show such a node to the user.

Thus, for example, if user **2** has an entry of 0000H for node **1**, user **2** will not see the icon for node **1** when navigating the Directory Service structure via the Explorer. (As described below, entries of 00000H are not actually stored.)

In the preferred embodiment, the access rights values of the access control matrix **300** are generally in the form of privilege level masks, with each defined bit corresponding to a respective user privilege level. FIG. **3B** illustrates a preferred basic set of user privilege levels, and the correspondence between these privilege levels and the bits of the access rights values. With reference to FIG. **3B**, bits **0–6** correspond respectively to the user privilege levels of viewer, observer, user, host, sysop manager, sysop and supersysop, and bits **7–15** are reserved for future definition. Thus, for example, an access rights value of 0024 hexadecimal (bits **2** and **5** set to one, and all others clear) indicates user privilege levels of "sysop" and "user."

Although this approach uses a hierarchy of privilege levels, various non-hierarchical approaches are possible. For example, the access rights values may directly specify the access operations that can be performed by the users, with, for example, bit **0** specifying whether the user has read-only access, bit **1** specifying whether the user has read/write access, and so on.

In the preferred embodiment, the general privilege levels of FIG. **3B** are transformed into specific access capabilities by the various on-line services (such as Chat, BBS, and the Directory Service). For example, the Chat service may give moderator-type access capabilities to users that have the privilege level of "host." The access capabilities corresponding to a given privilege level may vary from on-line service to on-line service. Generally, however, the access capabilities within a given on-line service will be consistent with the following privilege-level "definitions":

Viewer. The user can see the existence of the node, but cannot open or access the corresponding service. The user may be given the ability to subscribe to the service (to obtain a higher privilege level with respect to the service), and may be able to view certain (such as a textual description) properties of the node. (This is the lowest level of access rights a user can have with respect to a node. A user with no access rights with respect to a node cannot view the name, icon, or any other feature of the node).

Observer. The user can see the existence of the node and can open the service, but cannot actively participate in the service. (For example, an observer for a BBS folder node may be given read-only access to the messages within the folder). The user may be given the ability to subscribe to the service.

User. The user can do whatever is "normal" for the particular service. For example, the user may be given the ability to post BBS messages within public BBS folders, or may be given the ability to actively participate in public Chat conferences.

Host. The user is given host-level or leadership-level privileges (where applicable) for the service. For example, the Chat service may give the host user moderator privileges.

Sysop. The user is given the access rights consistent with normal (entry-level) sysop-type activities for the service, such as the ability to delete BBS messages, or the ability to edit a certain subset of the properties of a node.

Sysop Manager. The user is given various ownership-type privileges with respect to the node. For example, the sysop manager for a given node may be given the ability to change any of the properties (e.g., name, icon ID, etc.) for that node.

Supersysop. The user has the highest level of access authority provided by the service.

As indicated by the foregoing, the privilege level definitions are generally open ended, giving the various services flexibility in assigning specific access capabilities to users. This is particularly true for the privilege levels of "user," "host," and "sysop," which may be translated into significantly different access capabilities by different services.

Advantageously, the privilege levels are not limited to predefined accesses capabilities such as read-only, read/write, modify, append and delete, but rather are flexible enough to include new types of access capabilities that may later be defined. Thus, as new types of access capabilities are defined (when, for example, new services and new object types are created), these new access capabilities can be implemented using the existing user privilege levels. In other embodiments of the invention, the access rights values may correspond uniquely to predefined sets of access operations.

By way of example, suppose that a voice-based Chat service is added which assigns a "voice override" priority level of either low, medium or high to each member of a given voice Chat conference, to thereby give certain users a greater degree of control over the conversation than others. To implement these three newly-defined access capabilities without defining new user privilege levels, bits **2**, **3** and **5** in FIG. **3B** (corresponding to user privilege levels of user, host and sysop) could be used, respectively, to specify voice override priority levels of low, medium and high.

With further reference to FIG. **3B**, additional user privilege levels can be defined as needed (using bits **7–15**) to achieve higher degrees of privilege-level granularity. Also, services can be configured to give special meaning to certain combinations of privilege level bits. For example, an on-line service could give special access capabilities to users that have both the "host" and "sysop" bits set.

To simplify the description which follows, the term "access rights" will hereinafter be used to refer generally to the access rights values, and to the privilege levels and/or access capabilities associated with these access rights values.

With reference to FIG. **3A**, in a network that has on the order of millions of subscribers and thousands of content objects, the access control matrix **300** will be extremely large, and will normally exceed the virtual memory capacity of conventional servers. Thus, it would not be feasible to store the entire access control matrix **300** on a single server. Further, even if the access control matrix **300** were divided and stored across multiple servers, the time required to search the access control matrix **300** (to determine the rights of a user with respect to an object) would be long, and the user would therefore experience significant time delays when moving from object to object.

In accordance with the present invention, the above-described limitations are overcome by effectively compressing the access control matrix **300** both horizontally and vertically, to thereby reduce the quantity of access rights data that needs to be stored. Horizontal compression (the reduction of the number of columns) is effectively achieved by grouping together content objects which may be treating the same for security purposes. Vertical compression (the reduction of the number of rows) of the matrix **300** is effectively achieved by the formation of user groups. Each compression technique is described in detail below. The

compression of the access control matrix **300** is "effective," rather than actual, since the access control matrix **300** is not ordinarily generated in the uncompressed form of FIG. **3**A.

The compression of the access control matrix **300** advantageously enables the information contained therein to be stored on each security server **150** (within each relational access rights database **152**).

5. Compression by Grouping of Objects (FIGS. **4**A and **4**B)

In accordance with the present invention, the number of columns of the access control matrix **300** is reduced by effectively grouping together the content objects that can be treated the same for security purposes, and then storing only the access rights information for each group (rather than each content object). With reference to FIG. **4**B, each object group is identifiable by a mnemonic name ("Internal Public," "Internal 18-and-older," etc.), but is represented by a unique, 32-bit value referred to herein as the "security token" (or simply "token"). For example, the object group with the name "Internet 18-and-older" has a security token of 4(i.e., 0004H). The mnemonic names generally represent different content categories that have been defined for security purposes. To help to distinguish object groups from user groups (which are discussed below), the term "content categories" will be used herein to refer to the object groups. The security tokens serve as content category identifiers.

With reference to FIG. **4**A, which illustrates the horizontally compressed access control matrix **300'**, each column of the matrix corresponds to one content category, and is represented by the content category's security token. Because the total number of content categories will normally be significantly lower than the total number of content objects, the number of columns will be significantly reduced over the access control matrix **300** of FIG. **3**A.

With reference to FIG. **4**B, each content category (i.e., object group) contains the content objects which fall within a predetermined security classification. For example, the category "Internet 18-and-older" contains all Internet objects which have been classified (typically by system administrators) accordingly. In the preferred embodiment, each content object (or equivalently, each node of the Directory Service structure) is assigned to exactly one content category, and a content category can contain as few as one content object.

As described below, security tokens are also preferably defined for certain non-Directory Service entities, such as distribution lists for sending electronic mail, and connections to classes of services. This allows the GetAccoutRights API and access rights database **152** to be used to control access to entities that do not correspond to respective nodes of the Directory Service. These non-Directory-Service security tokens are not stored as node properties, but rather are stored by the entities (such as the Mail servers **120** and Gateways **140**) with which they are associated.

As a result of the grouping of the content objects, a user's privilege level (or privilege levels) will be the same for all content objects within a given content category. For example, if a given user has the privilege level of "observer" with respect to one content object in the Internet 18-and-older content category, that user will also have the privilege level of observer with respect to all other content objects of the Internet Public content category.

Although the categories listed in FIG. **4**B are content based, other bases for categorizing the data entities to which access is controlled are possible. For example, in embodiments of the invention that involve the control of accesses to software resources, the data entities may be grouped according to resource types, with categories such as "user-level threads," "system-level threads," "executable files," and "semaphores."

With further reference to FIG. **4**B, the content categories corresponding to tokens **1–4** are a basic set of groups which may be used with an initial implementation of the network **100**. As the content of the network grows, these content categories may be subdivided into sub-categories, to thereby achieve a higher degree of access rights granularity with respect to different types of content objects.

The content categories corresponding to tokens **100** and **101** are examples of content categories which may be defined to provide privacy over certain types of data. The content category "Corporation X Beta Test Data," for example, may contain all BBS objects (e.g., folders and messages) which pertain to the beta test of a software product of Corporation X. Access to such a group could be limited to individuals who are participating in the beta test, plus certain employees of Corporation X. This would allow Corporation X to privately correspond with beta test participants, without other users being able to view such correspondence. The content category "Family and Friends for Brown Family" may similarly be formed to allow private correspondence between a small group of subscribers (e.g., Brown family members plus designated friends), and may contain, for example, Chat and BBS objects which have been designated for this purpose. Of course, many different family and friends content categories can be defined to permit private correspondence between many different subgroups of users.

As indicated above, the 32-bit security tokens are preferably stored by the Directory Service as properties of nodes (in addition to certain tokens that have been defined for controlling access to non-Directory-Service entities). With reference to FIGS. **2** and **4**B, for example, nodes **6**, **7** and **8** could each have the security token of 0002H stored as a property, indicating that the three corresponding content objects are classified as "Internal 18-and-older" data for security purposes. As described in the following sections, the storage of the security tokens as node properties permits the Directory Service to rapidly and efficiently determine the rights of a user at a particular node. In other embodiments of the invention, the security tokens may be stored elsewhere within the system. For example, each on-line service could store or cache the security tokens for its own content objects.

In the preferred implementation of the network, security tokens are defined by system administrators as needed, and are entered as properties of nodes (typically only by users with at least sysop-level privileges with respect to such nodes) using the Sysop Tools client application. When, for example, a new service is created on the network, the new service can either be assigned its own security token (to allow separate security for the area), or can use an existing security token, such as the security token of the new service's parent node.

6. Compression by Grouping of Users (FIGS. **5**A and **5**B)

In the preferred embodiment of the on-line services network **100**, large numbers of users will typically have the same or similar access rights with respect to many of the content objects. Thus, the number of rows of the access control matrix **300'** can be significantly reduced by assigning users that have like access rights to user groups, and by storing the access rights of the user groups in place of user-specific access rights. In the preferred implementation of the network **100**, this technique reduces the number of rows by several orders of magnitude.

FIG. **5**A illustrates the access control matrix of FIG. **4**A following the assignment of users to user groups, and FIG. **5**B illustrates a preferred basic set of user groups. Each user group is identifiable by a mnemonic name ("everyone,"

"allsysops," etc.), but is represented internally by a 16-bit group ID. Each user group represents a group of users (i.e., user accounts). The following is a brief description of the basic user groups listed in FIG. 5B.

Everyone. All user accounts.

AllSysops. All users that have sysop privileges with respect to at least one content category. Members of this group can use the Sysop Tools client application to edit the Directory Service structure (FIG. 2), although the specific capabilities of users to edit the structure will normally vary from user to user.

SuperSysops. A small group of system administrators that have generally unlimited accesses rights. Supersysops can, for example, define new user groups and new security tokens.

Guest. User accounts that are used for demonstrations and marketing purposes.

Registration/signup. Accounts that are limited to registration and signup privileges.

18-and-older. Accounts which have access to 18-and-older-only type content objects.

Other groups may include, for example "Company X Beta Test Users," "Company Y Employees," etc.

Associated with each user group is a corresponding set of access rights, which are specified by a respective group-specific row **502** of the access control matrix **300"**. The access rights for group 1 (i.e., the group "everyone," which has a group ID of 1), for example, are represented by the access rights values in the first row of the matrix **300"**. The total number of user groups (and thus the number of group-specific rows **502**) will typically be very small in comparison to the total number of users. For example, in a network with millions of users, the number of user groups may be as few as several hundred.

Every user account (and thus every user) is assigned to at least one, and possibly multiple user groups. When a user is a member of multiple user groups, the user has all of the access rights associated with both such groups. For example, if a user is a member of both the "everyone" group and the "allsysops" group, the user will have all of the access rights associated with the everyone group plus all of the rights associated with the all sysop group.

In the preferred embodiment, user groups are defined by system administrators based on need. Membership within each group is controlled by updating a group-member table **602** which is stored on the security servers **150**. The group member-table **602** contains the user group IDs and corresponding user account numbers for every user group that has been defined. Updates to the group-member table **602** can be made by system administrators using a database editing program. Updates to this table can also be made automatically in response to certain user actions. For example, a service which provides an on-line subscription feature may be configured to automatically update the group-member table **602** whenever a user subscribes to the service, to thereby add the user to a corresponding user group. The group-member table **602** is further described below under the heading ACCESS RIGHTS DATABASE.

With reference to FIG. 5A, in addition to the rows **502** that specify the access rights of the various user groups, the compressed access control matrix **300"** includes account-specific rows **504**. Each account-specific row specifies rights that are to be "added on"to the group-based rights of the corresponding user. For example, if user A (FIG. 5A) is a member of user groups **1** and **2** only, the access rights of user A will be the rights of group **1**, plus the rights of group **2**, plus the rights specified in the account-specific row for user A. When the rights of a given user to a given content category are specified in multiple rows of the compressed access control matrix **300"**, the associated access rights values (XXXX) are logically ORed together to produce a summation of the row-specific rights. By way of example, suppose that user A has certain access rights with respect to content category **1** (i.e., the content category corresponding to token **1**) by virtue of being in user groups **1** and **2**, and that user A has also been given special rights (such as sysop privileges) with respect to content category **1** that are specified in the account-specific row for user A. To generate a 16-bit access rights value for user A with respect to content category **1**, the three 16-bit access rights values (group **1**, token **1**), (group **2**, token **1**), and (user A, token **1**) are logically ORed together. Numerical examples of this process are provided below.

As indicated by the foregoing, the account-specific rows are used to give certain users "special" access privileges beyond the "general" or "group-based" access privileges obtained by virtue of being in one or more user groups. For example, an account-specific row may be added to give a particular user sysop privileges with respect to a certain BBS folder and its contents, or to give the user moderator privileges with respect to a particular Chat conference. Typically, the number of account-specific rows **504** of the compressed access control matrix **300"** will be very small in comparison to comparison to the total number of users of the network **100**. For example, for a network having millions of users, the number of account-specific rows **504** will typically be in the hundreds.

In the preferred implementation of the access rights database **152**, which is described below, further compression of the compressed access control matrix **300"** is effectively achieved by storing only the non-zero entries (i.e., access rights values not equal to 0000H) of the matrix.

7. Access Rights Database (FIG. 6)

FIG. **6** illustrates a preferred implementation of the access rights database **152**. Generally, the access rights database **152** includes all of the information represented by the compressed access control matrix **300"**, plus a table **602** that indicates the members (i.e., users) of each user group. As indicated above, the access rights database **152** is preferably stored on each security server **150**. In other embodiments, the access rights database **152** may be implemented elsewhere within the network. For example, the access rights database **152** could be implemented on one or more of the application servers **120** and/or Gateways **140**. Also, although the access rights database **152** is preferably implemented as a relational database, other database arrangements are possible. For example, a hierarchical database could be used.

In the preferred embodiment, the access rights database **152** is generated and updated directly, without initially generating and/or compressing an access control matrix **300**. Stated differently, the above-described horizontal and vertical data compression techniques are inherent features of the preferred database implementation. It is contemplated, however, that these compression techniques can be used to transform an existing access rights database (such as a database of an existing network) into a relational database of the general type shown in FIG. **6**.

With reference to FIG. **6**, the access rights database **152** includes three tables: a group-member table **602**, a group-token table **604**, and an account-token table **606**. The group-member table **602** specifies the membership of each user group that has been defined, with each row of the table **602** specifying one user group and one user who is a member of

the group. User groups are specified in the table **602** by their 16-bit user group IDs, and users (i.e., user accounts) are specified by their 32-bit user account numbers. With reference to the example table entries shown in FIG. **6**, user group **1** includes at least users **1** and **2**, and user group **2** includes at least users **2** and **27**.

The group-token table **604** corresponds to the group-specific rows **502** (FIG. **5**A) of the compressed access control matrix **300**". Each row of the group-token table **604** specifies one user group, a content category (specified by its 32-bit security token) to which members of the user group have access, and the access rights (in the form of privilege levels) the group's members have with respect to the objects of the content category. By way of example, the first row of the group-token table **604** indicates that members of group **1** have access rights of 0004H (specifying a privilege level of "user," as indicated by FIG. **3**B) with respect to all objects within content category **5**. The account-token table **606** corresponds to the account-specific rows **504** (FIG. **5**A) of the compressed access control matrix **300**". Each row of the account-token table **606** specifies one user (i.e., one user account), a content category to which the user has access, and the account-specific access rights the user has with respect to objects within that content category. By way of example, the first row of the account-token table **606** indicates that user **1** has access rights of 0008H (indicating the privilege level of "host") with respect to objects within content category **5**. These account-specific access rights are in addition to the group-based rights of 0004H that user **1** has with respect to content category **5**. Thus, user **1** will be given both user-level (0004H) and host-level (0008H) access capabilities with respect to all objects within content category **5**.

Updates to the tables **602**, **604**, **606** are preferably made by system administrators using a database editing program which is part of the Sysop Tools client application. As will be appreciated by those skilled in the art, any of a variety of conventional database editing packages may be used for this purpose.

In addition to or in place of the account-token table **606**, an exclusion table (representatively shown by the account-token table **606**, which is identical in format) may optionally be implemented to take away certain group-based rights of users. The exclusion table has the same format as the account-token table **606**, but specifies the access rights that are to be subtracted from (or "masked off") the user's account, with respect to the content category specified therein. For example, an exclusion table row containing the entries (account no.=2), (token=5), (access rights value=0020H) would indicate that the group-based access rights of 0020H are to be masked off from the account of user **2**, leaving user **2** with access rights of only 0004H with respect to content category **5**. (Without this exclusion table entry, the rights of user **2** with respect to content category **5** would be 0024H, indicating "sysop" and "user" level privileges.)

The implementation of an exclusion table is useful, for example, for taking away access rights of users who misuse the on-line services. For example, if a particular user consistently uses profanity in BBS messages, the exclusion list could be used to lower that user's BBS access capabilities to a read-only level. The handling of exclusions on an exception basis advantageously permits the benefits of compressing the access rights matrix to be retained.

As will recognized by the foregoing, the inclusion of either an account-token table or an exclusion table will advantageously allow access rights to be customized on a peruser basis.

8. queries of Access Rights Database (FIGS. **7** and **8**)

In the preferred embodiment, each security server **150** is programmed to receive account-specific access rights queries from the application servers **120** and Gateways **140** within the network, and to respond to each such query by returning all of the access rights data of the user specified in the query. The queries are in the form of remote procedure calls (RPCs) which specify the account number of a single user, and are generated by the calling servers (i.e., the application servers **120** and Gateways **140**) using a GetAccountRights API. A round robin approach is preferably used to assign specific queries to specific security servers **150**.

To reduce the frequency of queries to security servers **150** (and to avoid the delay associated with such queries), the GetAccountRights API implements a caching scheme wherein the user-specific access rights data returned by the security server **150** is stored within an access rights cache **802** (FIGS. **8** and **9**) of the calling server. The GetAccountRights API and associated caching scheme are described below. FIG. **7** illustrates the sequence of steps taken a security server **150** each time a query is received for the access rights of some user (designated as "user X" in FIG. **7**). With reference to block **702**, the group-member table **602** is initially accessed to identify all of the user groups of which the user is a member. If the subject of the query is user **2** (FIG. **6**), for example, this step would identify groups **1** and **2** (and any other groups in which user **2** is a member).

With reference to block **704**, once the user groups have been determined, the group-token table **604** is used to identify the content categories (identified by their respective security tokens) to which the user has access, and to obtain the access rights values corresponding to such content categories. If the user has multiple access rights values corresponding to the same token (by virtue of being in multiple user groups), these access rights values are logically ORed together to produce a single 16-bit access rights value, as generally described above. Assuming for purposes of example that the entries shown in FIG. **6** are the only table entries, this step would produce the following results, respectively, for users **1**, **2** and **27**:

USER **1**:
Token **5** rights=0004H
Token **9** rights=0001H
USER **2**:
Token **1** Rights=0004H
Token **5** rights=(0004H) OR (0020H)=0024H
Token **9** rights=0001H
User **27**:
Token **1** Rights=0004H
Token **5** rights=0020H

The result of the step of block **704** is a group-based access rights list, which specifies the access rights (in the form of tokens and corresponding access rights values) the user has by virtue of being a member of one or more user groups. These access rights are referred to herein as the user's "group-based" access rights.

With reference to block **706**, once the user's group-based access rights have been obtained from the group-token table **604**, the account-token table **606** is accessed to obtain any additional rights that are to be added to the user's group-based rights. For user **1**, for example, token **5** rights of 0008H and token **6** rights of 0001H would be obtained. Since user **1** already has group-based rights of 0004H with respect to token **5**, the access rights values 0004H and 0008H will eventually be ORed together to produce a single 16-bit value. As described below, this step of ORing the

group-based and account-based access rights values is performed, if at all, by the calling server after the query returns.

In embodiments of the access rights database **152** that include an exclusion table, the exclusion table is then accessed to obtain any access rights that are to be taken away from the user's account. This step is similar to the step of accessing the account-token table **606** (since the two tables are identical in format), except that any access rights values read from the exclusion table are applied as masks for masking off the group-based rights specified therein. The step of masking off the user's rights can be performed either before or after the query returns.

The result of the steps **704** and **706** is an access rights list which contains all of the tokens and corresponding access rights values for the user. For user **1**, for example, the access rights list would have the following entries (assuming no other table entries exist):

(T5, 0004H), (T9, 0001H), (T5, 0008H), (T6, 0001H).

Each entry in this list is in the form of a 32-bit token (designated by the letter "T") followed by the corresponding 16-bit access rights value. Tokens which do not appear in this list (such as Token **7**) represent content categories to which the user has no access rights, and correspond to content objects which will not be shown to the user by the Directory Service. As illustrated for token **5** in this example, an access rights list may have two entries for the same token, since the account-specific access rights values are kept separate from the group-based values.

In other embodiments of the invention, the access rights values may be omitted from the access rights lists, so that each user-specific access rights list consists simply of a string of security tokens (i.e., category identifiers) that identifies the content categories to which the user has access. This may be desirable, for example, in systems that do not require the specification of access rights on a per-object (or on a per-object-category) basis.

With reference to block **708**, once the full access rights list for the user has been generated, the security server **150** sorts the tokens in numerically ascending order. For the user **1** access rights list shown above, this step may render the following list:

(T5, 0004H), (T5, 0008H), (T6, 0001H), (T9, 0001H).

With reference to block **710**, this list is then returned to the calling server. The calling server stores this user-specific access rights list within its access rights cache **802**, and searches this list for specific tokens to determine the access rights of the user with respect to specific content categories. As described below, the step of numerically sorting the tokens facilitates cache searches by the calling server for specific tokens.

FIG. **8** illustrates the process by which a Directory Service server **120** (i.e., a DirSrv or BBS server) queries a security server **150** to determine the access rights of a user, user X, and illustrates the caching scheme used by the Directory Service server **120** to cache access rights data. The query is in the form of an RPC call to the security server **150**, specifying the account ID of the user. This query will typically be generated when user X initially opens the Explorer window.

The security server **150** responds to the query by accessing its locally-stored copy of the access rights database **152**, and by returning the entire numerically-ordered access rights list for user X. This access rights list specifies the access rights of user X will respect to all nodes of the Directory Service structure. The Directory Service server **120** stores the user's access rights list within a user-specific row of its access rights cache **802**. As user X subsequently moves through the Directory Service structure (FIG. **2**) to view the various content objects, the Directory Service server **120** checks the cache row corresponding to user X (provided that the row has not been flushed from the cache) for the security tokens of the various nodes of the Directory Service structure. As described above, these security tokens are preferably stored as node properties of the Directory Service structure.

Each check of the cache **802** is initiated by the Directory Service by generating a GetAccountRights call, specifying as parameters of the call the user's account number and a token. The GetAccountRights API either returns the 16-bit access rights value of the user with respect to the token (i.e., with respect to the node which has the token stored as a property), or else returns a code indicating that the user does not have access with respect to the token. If no row exists in the cache **802** for the user, the GetAccountRights API generates a query to a security server **150** to create a cache row for the user, and then checks the cache row for the specified token. If a cache row is already present for the user, no query is necessary, since the information stored in the user's cache row fully specifies the user's access rights with respect to all nodes of the Directory Service structure.

To provide a specific example, suppose that a user double clicks on the icon for node **6** (FIG. **2**) of the Directory Service structure (assuming that the Directory Service has already returned the properties of node **6**). The Directory Service will respond by reading the security tokens for nodes **7** and **8** (which are stored as properties of these nodes), and by generating two GetAccountRights calls, one for node **7** and one for node **8**. The GetAccountRights call for node **7** will result in a check of the user's cache row (and possibly a query to create the cache row) for the token corresponding to node **7**, and the GetAccountRights call for node **8** will result in a check of the user's cache row for the token corresponding to node **8**. Each GetAccountRights call will return with either a 16-bit access rights value, or, if the security token is not found in the user's access rights list, a code indicating that the user does not have access to the node. If the user does not have access to the node, the Directory Service does not show the node to the user, and the user is prevented from either seeing the node or accessing the corresponding content object continuing the above example, if the user's access rights list (stored in the cache **802**) does not contain the security token for node **7**, the GetAccountRights API will return a code indicating that the user cannot access node **7**. The Directory Service will respond to this code by not sending any node **7** properties to the user's computer **102**, so that the node will not be displayed by the Explorer to the user. The user will thereby be prevented from accessing node **7**. If, however, the token for node **7** is found in the user's cache row, the GetAccountRights API will read the corresponding 16-bit access rights value from the cache **802**, and will return this value to the Directory Service. The Directory Service will then return the properties of node **7** that were requested by the Explorer (as parameters of a GetChildren call, as described above), and the Explorer will display node **7** to the user. The Directory Service and/or Explorer may additionally perform certain actions based upon which of the privilege level bits are set in the 16-bit access rights value. For example, if the "sysop manager" privilege level bit (bit **4** in FIG. **3B**) is set, the Directory Service will inform the Explorer (upon request

from the Explorer) that the user has sysop manager privileges at node **7**, and the Explorer will display a Sysop Tools edit menu that allows the user to edit the properties of node **7**. The GetAccountRights API is described in further detail below.

During a typical logon session, the Directory Service will request the user's access rights to hundreds of different content objects, and will thus generate hundreds of GetAccountRights calls. As described below, only the first GetAccountRights call for the user will result in a query to a security server **150**, and all subsequent GetAccountRights calls will normally result in a check of the user's row in the cache **802** without a new database query. Because accesses to the local access rights cache **802** are typically much faster than queries of the network-wide access rights database **152**, use of the access rights cache **802** significantly increases the performance of the GetAccountRights API, and thereby allows the user to rapidly move from node to node of the Directory Service structure. The storage of the security tokens as Directory Service node properties provides for a high degree of performance of the Directory Service, which is the service which typically generates the most GetAccountsRights calls.

Although the description thus far has focussed on the use of the GetAccountRights API by the Directory Service, as noted above, other services and machines on the network **100** can also preferably use the API to determine the rights of users. For example, Chat servers **120** may generate GetAccountRights calls as a user moves from Chat object to Chat object within the Chat service. The process by which a non-Directory-Service machine determines the access rights of a user to an object is generally the same as shown in FIG. **8** and described above.

Further, although the foregoing description has focussed on the security tokens that are stored as properties of Directory Service nodes, as indicated above, security tokens may also be stored by other types of entities, so that security can be provided via the GetAccountRights API without creating a corresponding Directory Service node. In the preferred embodiment, for example, the Mail servers store security tokens in association with mail distribution lists, and use these tokens (and the GetAccountRights API) to determine whether individual users are authorized send mail via such distribution lists. Also, the Gateways **140** store security tokens which correspond to various classes of services, including a class of generally-available services, a class of public services that are made available to the general public (e.g., non-subscribers), and a class of toll free services. Whenever a user requests to connect to a service, the corresponding Gateway **140** calls GetAccountRights (using the token of the corresponding class), and opens a pipe to the service only if the user is authorized to access the service.

In the course of a typical logon session, a user may connect to many different application servers **120** and services, and may access many different content objects within each service. Thus, the access rights list of the user will typically be cached on multiple different machines **120**, **140** within the network at the same time.

In the preferred embodiment, if an update is made to a user's access rights (via an update to the relational database **152**) while the user's access rights list is cached on a machine **120**, **140**, the cached access rights list will continue to be used, even though it is no longer up-to-date. Thus, an update to the user's access rights will not take effect until the next time the user's access rights list is read from the database **152**. In other embodiments, a mechanism may be provided for invalidating all cached copies of a user's access rights list upon the occurrence of certain events, such as upon the generation of an exclusion table entry for the user.

9. Access Rights Cache (FIG. **9**)

FIG. **9** illustrates a preferred implementation of the access rights cache **802**, as implemented on the application servers **120** which place GetAccountRights calls. The cache **802** contains 5000 rows, and can thus hold the access rights lists of 5000 different users. Because the number of user-specific service sessions handled by a given application server **120** normally will not exceed 5000, the cache **802** is large enough to hold all of the access rights information of all users who are being serviced by the application server **120**. For the Gateways **140**, an access rights cache of 1000 rows is used, since each Gateway can handle a maximum of 1000 simultaneous user connections. In other embodiments, the number of cache rows per machine **120**, **140** may be allocated dynamically, with the maximum number of cache rows per machine depending upon the amount of memory available on each respective machine.

Each row of the cache **802** contains 500 slots. Each slot stores a 32-bit security token and the corresponding 16-bit access rights value. Each user-specific row can thus store an access rights list having a length of up to 500 tokens, which is sufficient to fully specify the access rights of the user with respect to all nodes of the Directory Service Tree. (Because many nodes will typically have the same security token, the number of nodes to which the user has access may greatly exceed 500.)

The cache **802** is preferably implemented in the dynamic RAM of each machine **120**, **140** that places GetAccountRights calls. As described above, multiple machines **120**, **140** may simultaneously cache the access rights data of the same user. For example, the user's access rights list may simultaneously be stored in the respective caches **802** of a DirSrv server **120** to which the user is connected, a Chat server to which the user is connected, and the Gateway **140** that is handling the user logon session.

In the preferred embodiment, only a single access rights cache **802** is implemented on any given machine **120**, **140** at a time, even if the machine is allocated to multiple service groups. Thus, for example, if two different service applications (such as the Chat and DirSrv applications) are concurrently running on the same application server **120** and both generate GetAccountRights calls, these two service applications will share the same access rights cache **802**.

With further reference to FIG. **8**, each machine which implements an access rights cache **802** contains cache flushing structures **806** which monitor certain activities to determine when a user-specific access rights list may be overwritten in the cache **802**. The first such structure is a least-recently-used (LRU) monitor **808** which monitors accesses to the cache rows to keep track of which row was least recently accessed. The LRU monitor **808** specifies, when the cache **802** is full (i.e., all rows occupied), the cache row that is to be overwritten when a new access rights list is returned by a security server **150**. Least-recently-used algorithms are well known in the art.

The second cache flushing structure is a pipe monitor **810** which monitors the number of pipes that each user has to the application of the application server **120** (or Gateway **140**) on which the cache **802** resides. Whenever the pipe monitor **810** detects that the number of pipes for a given user has gone to zero (indicating that the user has disconnected from the server **120**), the user's access rights list is deleted from the cache **802**.

When a cache row is created for a user, the slots of the user's cache row are filled in sequential order (from the

lowest slot number to the highest slot number) as the numerically-ordered access rights list is returned by the security server **150**. As illustrated in FIG. **9** for the example access rights list of user **1**, the tokens (and corresponding access rights values) are written to the cache **802** in numerically ascending order. As illustrated by slots **1** and **2** for user **1**, duplicate tokens may be present in the list, indicating that the user has been given additional rights via the account token table **606**. These duplicate tokens will always fall in adjacent cache slots.

To obtain the access rights of the user with respect to a given content category, the GetAccountRights API performs a binary search of user's cache row for the token specified as a parameter of the API. If the token is found, the corresponding access rights value (stored in the same cache slot as the token) is read from the cache **802**. The GetAccountRights API also checks the adjacent slot or slots for duplicate tokens. If a duplicate token is found, the corresponding access rights value is read and logically ORed with the first access rights value to generate a single 16-bit access rights value. By way of example, the call GetAccountRights (user **1**, token **5**) would cause the access rights values 0004H and 0008H to be read from the first two slots of the cache row for user **1**, and these two values would be ORed to produce 000CH.

Advantageously, the GetAccountRights API is structured to begin the binary search even if the cache row is currently being filled, allowing the API to return before the entire access rights list has been returned by the security server **150**. Using the access rights list for user **1** (FIG. **9**) as an example, if the search is for token **6**, and the cache row is currently being filled, it is possible (and likely) that token **6** will be found before the cache row for user **1** is complete. This feature of the GetAccountRights API increases performance on GetAccountRights calls which require a query of the access rights database **152**. To take full advantage of this feature, care is taken by system administrators to assign the lowest numbered tokens to the most commonly accessed object groups (since the lowest numbered tokens are the first to be returned by the security server **150**, and the first to be written to the cache). This feature of the GetAccountRights API is further described below.

10. GetAccountRights Method (FIG. **10**)

FIG. **10** illustrates the sequence of steps corresponding to the GetAccountRights API. These steps are performed by the application server **120** (or Gateway **140**) that generates the GetAccountRights call. As described above, the GetAccountRights API is called whenever it becomes necessary to determine the rights of a user with respect to a content object. The parameters of the GetAccountRights API are the 32-bit account number of the user (designated as "user X" in FIG. **10**) and the 32-bit token (designated a "token Y") of the node.

With reference to decisional block **1002**, the calling server **120** initially checks its access rights cache **802** to determine whether a cache row exists for user X. This is preferably accomplished using a conventional hash algorithm to search for the user's account number.

With reference to blocks **1004–1008**, if no cache row exists for user X, the server determines whether a query thread has been started to obtain the access rights of user X from the access rights database **152**. If a query thread has been started, the API sleeps for an appropriate interval (to allow a cache row to be created for user X), and then rechecks the cache **802**. If no query thread has been started, the API starts a query thread before sleeping and rechecking the cache **802**. The use of a separate query thread for creating and filling the user's cache row advantageously facilitates the concurrent filling of a cache row and searching of the cache row.

With reference to block **1012**, once a cache row has been created for user X (which may or may not be complete), a binary search is initiated for token Y. With reference to blocks **1014–1018**, as the binary search progresses, the API tests the results of the search and takes one of three actions. If the token is not found but the cache row is not yet complete (indicating that the access rights list is still being returned), the API sleeps and then retests the search results. If the token is not found and the user's cache row is complete, the API returns a code indicating that the user does not have any access rights with respect to the token.

With reference to blocks **1020–1022**, if token Y is found, the corresponding 16-bit access rights value is read from the cache. The API then checks the adjacent slot or slots in the cache for token Y. If a duplicate of token Y is found (indicating that user X has been given additional access rights with respect to token Y via the account-token table **606**), the corresponding access rights value is read from the cache and logically ORed with the first access rights value. The result of the logical OR operation is then returned.

11. Assignment of Tokens and Formation of User Groups

In the preferred embodiment of the network **100**, new security tokens are assigned by system administrators (to create new content categories) as it becomes necessary or desirable to provide separate security with respect to new or existing service areas. In accordance with one preferred mode of operation, security tokens are assigned so as to create service areas that are managed or "owned" by different individuals. The responsibility of monitoring and/or otherwise managing the content of the network is thereby be distributed among many (e.g., 500) different users, including system administrators, subscribers, and third party content providers.

To provide a specific example of how ownership may be assigned to service areas in accordance with the present invention, suppose that a system administrator wants to create a new service area, such as a bulletin board on a particular topic, and wishes to designate a particular subscriber as the owner of the new service area. (In the preferred embodiment of the network **100**, subscribers can request the creation of certain types of service areas, and can volunteer to be owners of such areas.) To generate the new BBS service area, the system administrator creates a BBS folder node (preferably using the Sysop Tools client application) in the Directory Service tree. To provide separate security for this new service area, the system administrator assigns a unique security token to the folder node, and enters this security token as a property of the folder node. (BBS messages subsequently created under the new BBS folder then inherit this security token, and become part of the same content category.) To give the user ownership-type privileges to the new service area, the system administrator then generates a user-specific row in the account-token table **606**, specifying (1) the user's account number, (2) the newly-created security token, and (3) an access rights value that has the "sysop manager" bit (bit **4** in FIG. **3B**) set. Finally, the system administrator adds one or more rows to the group token-table **604**, specifying in each such row: (1) the group ID of a user group which will be given access to the new service area, (2) the newly-created security token for the area, and (3) an access rights value that specifies the privilege level/s members of the group are to have with respect to the new service area. For example a row could be created to give members of group **1** (the group "everyone") user-level access to the new area.

In accordance with another preferred mode of operation, content categories and associated user groups are formed so as to create many different "private" service areas (such as the "family and friends" type service areas described above) that are accessible to different subgroups of users. To provide a specific example, suppose that a system administrator wants to create a Chat room to allow members of a certain organization to carry on an interactive conversation. To create such a Chat room, the system administrator initially creates a Chat room node, specifying a unique security token for the Chat room. The system administrator then updates the group-member table **602** so as to create a new group that consists of the accounts of the members of the organization. (If the group is small, the system administrator may forego creating a new user group, and may alternatively generate one user-specific row in the account-token table **606** for each member of the organization.) Finally, the system administrator adds a row to the group-token table **604**, specifying (1) the group ID of the newly-created user group, (2) the security token of the Chat room, and (3) an appropriate access rights value.

As will be recognized from the foregoing, content categories and user groups may be formed by system administrators to achieve any of a variety of different security-related objectives. These objectives will depend generally upon the nature of the particular network in which the present invention is employed, and will depend upon the type or types of data entities to which access is being controlled.

As will be apparent to those skilled in the art, the general criteria used by system administrators for deciding when to create new user groups and when to assign new security tokens will ultimately affect the quantity of data stored within the access rights database **152**. In the network **100** described herein, these decisions may advantageously be made as folder nodes are added to the Directory Service structure. The decision making process may be assisted or the decision may be made by a computer software system which monitors the contents of the access rights database **152**, and which recommends modifications that can be made to the existing user groups and content categories in order to reduce the quantity of data stored within the database **152**.

12. Other Embodiments

As described above, the preferred embodiment uses Directory Service nodes as the basic content unit with which different security levels may be associated. Thus, in order to provide security for a content object, a corresponding node (with a corresponding security token stored as a property) must be created in the Directory Service structure. As will be readily apparent to those skilled in the art, however, various alternatives to the node-based approach are possible. For example, the security tokens could be stored or cached with the content objects, or could be stored within tables maintained by the various services (such as Chat or Mediaview). (Accordingly, it will further be recognized that the present invention does not require the use of a directory structure). Hybrid approaches are also possible, in which the security tokens for some content objects (such as folder-type objects) are stored within a directory structure, while the security tokens for other content objects are stored elsewhere within the system.

It will also be appreciated that although the preferred embodiment described herein is directed to the security of user-accessible content objects in an on-line services network, other embodiments may be directed to the security of entirely different types objects and data entities. For example, the invention may readily be adapted to control user accesses to files in a file system, or to control accesses by software processes to system resources.

In view of these variations and other variations which may be apparent to those skilled in the art, the scope of the present invention is intended to be defined only by reference to the appended claims.

What is claimed is:

1. A method for controlling user access to a plurality of data entities in a computer network, said plurality of data entities stored on a plurality of application servers, said method comprising the steps of:

sending an access rights query from an application server to a security server, said access rights query specifying a user of the network;

at said security server, accessing a relational database in response to said access rights query to obtain an access rights list for said user, said access rights list specifying access rights of said user with respect to said plurality of data entities;

sending said access rights list from said security server to said application server;

at said application server, storing said access rights list in an access rights cache; and

accessing said cache to determine the access rights of said user with respect to a specific data entity of said plurality of data entities.

2. The method according to claim **1**, wherein said access rights list comprises a plurality of category identifiers, each of said category identifiers specifying a data entity category.

3. The method according to claim **2**, wherein said access rights list further comprises a plurality of access rights values, each of said access rights values corresponding to a respective one of said category identifiers and specifying access rights of said user with respect to data entities that fall within a respective data entity category.

4. The method according to claim **2**, further comprising the step of determining a data entity category in which said specific data entity falls.

5. The method according to claim **4**, wherein said step of determining a data entity category comprises accessing a directory structure which is stored on at least one of said plurality of application servers, said directory structure representing an arrangement of said plurality of data entities.

6. The method according to claim **4**, wherein said step of determining a data entity category comprises reading a category identifier stored with said specific data entity.

7. The method according to claim **4**, wherein said step of determining a data entity category comprises reading a category identifier that is stored on an application server in association with said specific data entity.

8. The method according to claim **2** wherein said step of accessing said cache comprises searching said cache for a specific category identifier, said specific category identifier representing a data entity category in which said specific data entity falls.

9. The method according to claim **2**, wherein said step of storing said access rights list in said cache comprises storing said category identifiers in a numerical order within said cache to thereby facilitate searches of said cache.

10. The method according to claim **1**, wherein said access rights list comprises a plurality of access rights values, said access rights values specifying generic privilege levels of said user.

11. The method according to claim **10**, wherein said step of accessing said cache comprises the steps of reading an access rights value from said cache, and translating said access rights value into a set of specific access capabilities.

12. The method according to claim **11**, wherein said step of translating is performed by a service application running

on said application server, said service application being associated with said specific data entity.

13. The method according to claim 1, wherein said step of accessing said relational database comprises identifying at least one user group in which said user is a member.

14. The method according to claim 13, wherein said step of accessing said relational database further comprises identifying a plurality of data entity groups to which said user has access rights by virtue of being a member of said at least one user group.

15. The method according to claim 1, wherein said step of storing said access rights list in said cache and said step of accessing said cache to determine the access rights of said user are performed concurrently.

16. The method according to claim 1, wherein said plurality of data entities represents the content of an on-line services network.

17. The method according to claim 1, wherein at least one of said plurality of data entities is a system resource.

18. The method according to claim 1, further comprising the step of forwarding said access rights list from said application server to a different application server when said user connects to said different application server.

19. The method according to claim 1, further comprising the step of, if said user is not authorized to access said specific data entity, preventing said user from seeing a representation of said specific data entity.

20. The method according to claim 19, wherein said step of preventing comprises omitting said representation from a reconstructed directory structure that is shown to said user.

21. A method of determining the access rights of a user of a computer system with respect to a plurality of data entities of the computer system, comprising the steps of:

identifying at least one user group of which said user is a member, said at least one user group being part of a predefined set of user groups; and

identifying at least one data entity category to which said user has access by virtue of being a member of said at least one user group, said at least one data entity category being part of a predefined set of data entity categories.

22. The method according to claim 21, wherein said steps of identifying at least one user group and identifying at least one data entity category each comprise accessing a relational database stored on a server of a computer network.

23. The method according to claim 21, further comprising the step of identifying at least one data entity that falls within said at least one data entity category.

24. The method according to claim 21, further comprising the steps of:

determining a specific data entity category in which a specific data entity falls; and

determining whether said at least one data entity category to which said user has access includes said specific data entity category, to thereby determine whether said user has access to said specific data entity.

25. The method according to claim 21, further comprising the step of reading an access rights value that specifies access rights of said user with respect to all data entities that fall within a data entity category of said at least one data entity category.

26. The method according to claim 21, further comprising the step of identifying at least one additional data entity category to which said user has access, said at least one additional data entity category being in addition to data entity categories to which said user has access by virtue of being a member of user group.

27. The method according to claim 21, wherein said step of identifying at least one user group of which said user is a member comprises identifying a plurality of user groups of which said user is a member.

28. The method according to claim 21, wherein each user group of said predefined set of user groups corresponds to a respective set of user access rights with respect to said plurality of data entities.

29. The method according to claim 21, wherein each data entity category of said predefined set of data entity categories contains a respective subgroup of said plurality of data entities.

30. The method according to claim 21, wherein each data entity of said plurality of data entities falls within exactly one data entity category of said predefined set of data entity categories.

31. The method according to claim 21, further comprising the steps of:

generating a list of category identifiers that identifies said at least one data entity category to which said user has access; and

transmitting said list across a computer network to at least one server.

32. The method according to claim 31, further comprising the step of storing said list in a cache memory of said at least one server.

33. The method according to claim 31, further comprising the step of storing said list in respective cache memories of a plurality of servers.

34. The method according to claim 21, wherein said plurality of data entities represents a content of an on-line services network.

35. The method according to claim 21, wherein said plurality of data entities comprises files of a file system.

36. The method according to claim 21, wherein said plurality of data entities comprises system resources to which access is controlled by an operating system.

37. In a computer network in which different users have different access rights with respect to different data entities, a method of efficiently specifying the access rights of users, comprising the steps of:

assigning each of a plurality of data entities to one of a plurality of categorical groups of data entities, each of said categorical groups containing data entities for which user access rights may be specified collectively; and

assigning each of a plurality of users to at least one of a plurality of user groups, each of said user groups having a corresponding set of access rights associated therewith with respect to said plurality of categorical groups.

38. The method according to claim 37, wherein said step of assigning each of said plurality of data entities to one of said plurality of categorical groups comprises storing a respective categorical group identifier in association with each of said plurality of data entities.

39. The method according to claim 38, wherein said step of storing comprises storing a categorical group identifier within a data entity directory structure.

40. The method according to claim 37, wherein said step of assigning each of said plurality of users to at least one of said plurality of user groups comprises assigning at least one of said users to multiple of said user groups.

41. The method according to claim 37, wherein each of said data entities is a content object that represents content of an on-line services network.

42. A system for providing user access to data entities in a computer network, comprising:

at least one application server that stores a plurality of data entities, said data entities accessible by a plurality of users through a plurality of application programs,

different of said users having different levels of access with respect to at least some of said data entities;

a database which stores access rights values that specify access rights of said users with respect to said data entities; and

an access rights cache on said at least one application server, said access rights cache storing access rights lists, said access rights lists obtained from said database in response to requests from said at least one application server, each of said access rights lists comprising a plurality of said access rights values and specifying access rights for a respective one of said plurality of users.

43. The system according to claim **42**, wherein said access rights values are stored in said database in association with category identifiers that identify categories of said data entities.

44. The system according to claim **43**, wherein each of said lists further comprises a plurality of said category identifiers.

45. The system according to claim **43**, wherein said database is implemented on a separate server from said at least one application server.

46. The system according to claim **45**, wherein said at least one application server stores at least a subgroup of said category identifiers.

47. The system according to claim **43**, wherein said access rights values are stored in said database in further association with group identifiers that identify groups of said users.

48. The system according to claim **42**, wherein said at least one application server runs a program module that generates a query of said database when a user connects to said at least one application server, said query causing an access rights list for said user to be obtained from said database and written to said access rights cache.

49. The system according to claim **48**, wherein said program module deletes said access rights list from said cache when said user disconnects from said at least one application server.

50. The system according to claim **42**, wherein said access rights cache specifies access rights for a variable subset of said plurality of users.

51. The system according to claim **42**, wherein each of said access rights lists specifies user access rights with respect to all of said data entities.

52. The system according to claim **42**, wherein said at least one application server comprises an application server that runs a directory service application program, said directory service application program providing a directory of said data entities to said users.

53. The system according to claim **42**, wherein said access rights values contain privilege level bits which specify general privilege levels, said general privilege levels converted into specific access capabilities by said application programs, different application programs converting like privilege levels into different access capabilities.

54. An access rights list stored on a storage medium of a computer, said access list specifying the access rights of a user of a network with respect to a plurality of data entities of said network, said plurality of data entities subdivided into multiple categorical groups of data entities, said access rights list comprising:

a plurality of group identifiers, each of said group identifiers specifying one of said multiple categorical groups, said plurality of group identifiers specifying a subset of said multiple categorical groups to which said user has access rights; and

a plurality of access rights values, each of said access rights values specifying access rights with respect to data entities which fall within a respective one of said categorical groups of said subset.

55. The access rights list according to claim **54**, wherein said group identifiers are arranged in a numerical order to facilitate searches for individual group identifiers.

56. The access rights list according to claim **54**, wherein said plurality of data entities represents content of an on-line services network.

57. The access rights list according to claim **54**, stored within an access rights cache of a server.

58. The access rights list according to claim **54**, stored within an access rights cache of a gateway computer.

59. A relational database for storing access rights data which specifies access rights of users with respect to a plurality of data entities of a computer network, said plurality of data entities subdivided into a plurality of categories, said database comprising:

a first table that maps users to user groups, at least one of said users being a member of multiple of said user groups;

a second table which contains, for each of said user groups, a group-based access rights list that specifies group-based access rights of members of a respective user group, said group-based access rights list stored in association with a plurality of category identifiers that identify said categories of data entities; and

a third table which contains, for a least one of said users, a user-specific access rights list that specifies special rights for a respective user, said user-specific access rights list stored in association with said plurality of category identifiers.

60. The relational database according to claim **59**, wherein said special rights are additional rights that are added to said group-based rights of said respective user.

61. The relational database according to claim **59**, wherein said special rights are exclusion rights that are subtracted from said group-based rights said respective user.

62. The relational database according to claim **59**, wherein said data entities are content objects of an on-line services network.

63. In a computer network in which different users have different access rights with respect to different data entities, a method of specifying the access rights of a user with respect to a plurality of data entities, comprising the steps of:

assigning a category identifier to said plurality of data entities;

storing said category identifier with or in association with each data entity of said plurality of data entities; and

storing an access rights value in association with said category identifier and in further association with an account number of said user, said access rights value specifying said access rights of said user with respect to said plurality of data entities.

64. The method according to claim **63**, wherein said access rights value comprises a plurality of privilege level bits, each of said privilege level bits corresponding to a respective privilege level which may be assigned to said user.

65. The method according to claim **63**, wherein said access rights value specifies a sysop privilege level of said user with respect to said plurality of data entities.

66. The method according to claim **63**, wherein said step of storing said category identifier comprises storing said category identifier in association with at least one node of a directory structure, said directory structure providing a directory to at least said plurality of data entities.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,941,947
DATED : August 24, 1999
INVENTOR(S) : Brown et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 13,
Line 39, "Dirsrv" should read -- DirSrv --.
Line 50, "Depending" should begin a new paragraph.

Column 23,
Line 18, "The" should begin a new paragraph.
Line 67, "peruser" should read -- per-user --.

Column 24,
Line 20, "FIG.7" should begin a new paragraph.

Column 26,
Line 47, "object continuing" should read -- object. Continuing -- with "Continuing" beginning a new paragraph.

Column 30,
Line 32, "thereby be" should read -- thereby --.

Signed and Sealed this

Twenty-third Day of December, 2003

JAMES E. ROGAN
Director of the United States Patent and Trademark Office